Moab/NODUS Cloud Bursting for Moab Workload Manager

Administrator Guide for Moab Workload Manager 9.1.3

and Amazon Web Services (AWS)

September 2018

© 2018 Adaptive Computing Enterprises, Inc. All rights reserved.

Distribution of this document for commercial purposes in either hard or soft copy form is strictly prohibited without prior written consent from Adaptive Computing Enterprises, Inc.

Adaptive Computing, Cluster Resources, Moab, Moab Workload Manager, Moab Viewpoint, Moab Cluster Manager, Moab Cluster Suite, Moab Grid Scheduler, Moab Grid Suite, Moab Access Portal, and other Adaptive Computing products are either registered trademarks or trademarks of Adaptive Computing Enterprises, Inc. The Adaptive Computing logo and the Cluster Resources logo are trademarks of Adaptive Computing Enterprises, Inc. All other company and product names may be trademarks of their respective companies.

Adaptive Computing Enterprises, Inc.
704 Goodlette Road North
Naples, FL 34102
+1 (239) 330-6083
www.adaptivecomputing.com

Scan to open online help

Contents

Welcome

Welcome to the *Moab/NODUS Cloud Bursting Administrator Guide for Moab Workload Manager 9.1.3* and Amazon Web Services (AWS).

This guide is intended for Moab/NODUS Cloud Bursting system administrators.

The following sections will help you quickly get started:

For pricing information, contact sales@adaptivecomputing.com.

For support, contact support@adaptivecomputing.com.

Chapter 1: Introduction to Moab/NODUS

This chapter provides high-level information about how Moab/NODUS Cloud Bursting works.

In this chapter:

1.1 Moab/NODUS Cloud Bursting Overview

Moab/NODUS is part of Moab Elastic Computing, which is an add-on package for Moab Workload Manager. Contact your Adaptive Computing account manager for more information.

Elastic Computing is only applicable for Torque Resource Manager and Native RMs with QoS triggers.

In this topic:

1.1.1 Bursting Environment

In order to burst to the cloud, the Moab Workload Manager (MWM), the Torque Resource Manger, and Moab Web Services (MWS) should be installed and configured. All three of these packages can be found on the *Adaptive Computing Download Center* (http://www.adaptivecomputing.com/support/download-center/), and the install documentation can be found in the *Moab HPC Suite Installation and Configuration Guide*, which can be found at the *Adaptive Computing Documentation Center* (http://www.adaptivecomputing.com/support/documentation-index/). Once these packages are installed, return to this documentation for steps on installing NODUS and configuring Moab to use it.

1.1.2 About Moab/NODUS Cloud Bursting

During the course of operation the number of job submissions will go up and down. Under some circumstances the job backlog may increase to the point where additional resources are required to complete the job backlog in a reasonable time frame. In this scenario, jobs will be held until resources become available. The Elastic Computing feature in Moab allows the Moab scheduler to take advantage of systems that can temporarily provide additional nodes (for example, to create new virtual machines or borrow physical nodes from another system) to fulfill the workload demand in a more timely manner.

Moab provides a general Elastic Computing framework that serves as a basis for Moab/NODUS Cloud Bursting, which can be configured to access multiple cloud providers either on-demand or based on a job backlog.

This chapter provides examples of the Elastic Computing and node end scripts. Your scripts will vary based on your system configuration. Contact your Adaptive Computing account manager for suggestions and options to configure Elastic Computing.

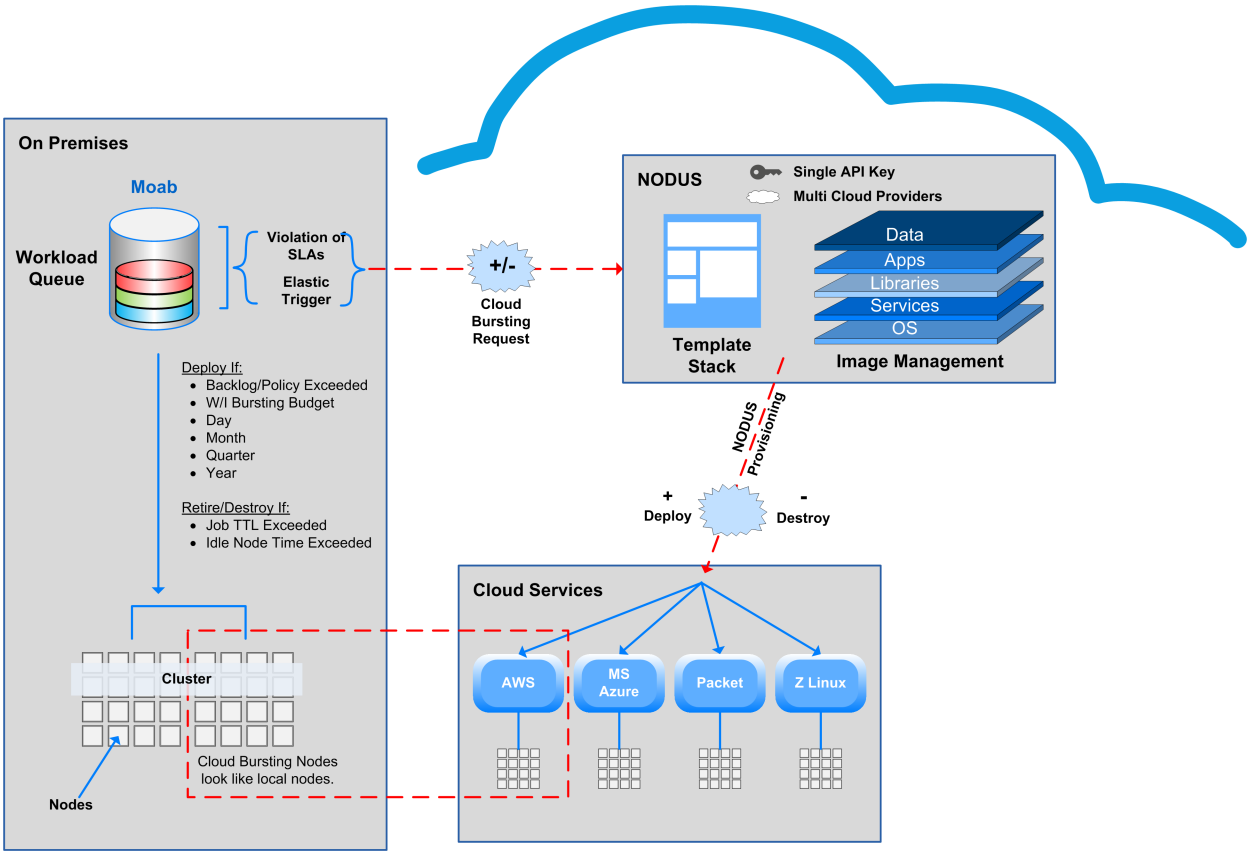
1.2 Moab/NODUS Cloud Bursting Architecture

Moab/NODUS Cloud Bursting is a configurable and extensible solution based on Moab's Elastic Com-puting framework, using the NODUS Platform to launch instances and build AMIs in your own Amazon Web Services (AWS) account. Elastic Computing with Moab/NODUS Cloud Bursting can either be triggered on-demand or by the size of the job backlog.

In this topic:

1.2.1 Backlog Bursting

The diagram below depicts Moab's Elastic Computing feature using NODUS:



Moab monitors the job backlog and, when a predefined threshold in the Moab config file is reached, fires the elastic trigger to call out to NODUS to launch a predefined number of instances in your AWS account. NODUS launches instances based on a prebuilt Amazon Machine Image (AMI) that

contains everything needed for jobs to run. See 3.3 Building Stacks with NODUS - page 23 for more information on setting up the AMIs (i.e. stacks).

The instances are added as nodes to Torque using the `qmgr` command. They are considered dynamic nodes because any number of nodes can be added. They are also given a time to live (`TTL`) that indicates each node's end-of-life. Access controls can also be added to reserve the nodes for specific users/groups. See 4.1 Dynamic Nodes - page 44 for additional information.

Moab can then begin scheduling jobs on the allocated nodes.

If dynamic nodes become idle for a specified amount of time (defined by the `NODEIDLEPURGETIME` parameter in the Moab config file) or when the node's time-to-live expires, Moab fires an "node end" trigger to remove the nodes from the Moab/Torque and calls out to NODUS to remove them from your AWS account.

1.2.2 On-Demand Bursting

Unlike backlog bursting, on-demand bursting is not triggered by the amount of work in the job backlog. Users can simply request that their job run in AWS by specifying the job template that is preconfigured in the Moab config file that contains the trigger defined to call out to NODUS. See 3.7 Configuring On-Demand Elastic Computing For Use With NODUS - page 39 for details.

1.2.3 Securing Data Transmissions

Because this solution allows for jobs to be run off-site, it is paramount that the transfer of data is done securely. There are three different ways to accomplish to create a secure connection from your network to your AWS account:

1. Create a point-to-point VPN connection from your network to a Virtual Private Cloud (VPC) in your AWS account. AWS does charge for this capability but this allows for one single, secure line of communication between your network and your AWS account. Transfer speeds will be based on your internet bandwidth and the distance from your network to the Region and Availability Zone you are bursting to. See VPN Connections in the AWS documentation for information on how this is to be configured.
2. Use AWS Direct Connect to configure a dedicated network connection from your network directly into your AWS account. Because it is a private connection that does not go over the public internet, there is no need for a VPN, and speeds between 1 Gbps or 10 Gbps are possible. This solution comes at a considerably higher price than the AWS VPN, but with advantage of a more predictable network speed. For more information see the AWS page on AWS Direct Connect.
3. Install a VPN client into the Amazon Machine Image (AMI) to connect to a VPN Server on your network when the AWS instance is launched. Although this solution requires the most up-front configuration, it allows the AWS instances to connect to your network securely with no cost from AWS. This solution uses one VPN connection per AWS instance, so there is a potential for excessive network overhead. Transfer speeds are also based on your internet bandwidth and the distance from your network to the Region and Availability Zone to which you are bursting.

Chapter 2: Configuring AWS for Cloud Bursting

To set up bursting from your HPC cluster, AWS requires some prior configuration. Further configuration is also possible but this document will discuss the minimum AWS configuration needed for bursting to work. This guide is not meant to be an instructional document on how to use AWS, but it will explain why certain EC2 configurations are needed. Other AWS services, such as RDS, DynamoDB, RedShift, etc. and solutions such as EBS volumes are not needed for this bursting solution, but could be could use if desired. For further information on AWS solutions, see the AWS documentation.

In this chapter:

2.1 AWS Bursting Policies

Without administrator access, certain policies must be created in AWS Identity and Access Management (IAM) in order to allow users and groups to have the privileges needed to build stacks and launch bursts. To create these policies, an AWS administrator can do the following:

1. Go to IAM > Policies > Create Policy.
2. Click on JSON and replace the contents with the JSON code found in `NODUS_Policies_Stack_Build.JSON` to create the policies needed for building stacks.
3. Name the policy "NODUS_Stack_Build" (the description is optional).
4. Click "Create Policy".

Repeat the process for `NODUS_Policies_Bursting.JSON` and name the policy "NODUS_Bursting."

After these two policies have been created, they can be applied to groups or tied directly to users to give the proper privileges needed to build stacks and launch bursts.

2.2 AWS Access and Secret Keys

In order to launch instances in your AWS account, you must generate an AWS Access Key ID and Secret Access Key. If your account is the AWS owner, the Access Key ID and Secret Access Key can be found under your account in "My Security Credentials > Access keys (access key ID and secret access key) > Create New Access Key." Be sure to save the Access Key ID and Secret Access Key by either downloading it or copying to a file, as it will be needed later to configure NODUS.

If your account is an IAM user with administrator access, the Access key ID and Secret Access Key can be found under "IAM > Users." Select your user, click on the Security Credentials tab, then click "Create Access Key." Be sure to save the Access Key ID and Secret Access Key, as it will be needed later to configure NODUS.

If your account is an IAM user without administrator access, an AWS administrator can generate the Access Key ID and Secret Access Key for you by following the same steps as an IAM user with administrator access above.

It is recommend that you rotate these keys on a regular basis by deleting the current key and creating a new one.

2.3 Create NODUS Key Pair

NODUS needs a `nodus` key pair in order to finalize the created instance. The `nodus` key pair can also be used to connect to the bursted instances. The following can be done on any machine that has the command *`ssh-keygen`* to create the key pair.

To create the NODUS key pair, perform the following steps:

1. Using the command *`ssh-keygen`*, create a new key pair, calling it `nodus`:

   ```
   % ssh-keygen -t rsa -C "nodus" -f ~/nodus
   ```

   This will create public/private key pair files named `nodus.pub` and `nodus` in your home directory.

2. Rename the `nodus` private key file from `nodus` to `nodus.pem`. This is to easily identify the private key file for future reference in this guide.

3. To add the `nodus.pub` public key to AWS, do the following:

   a. In the AWS console, navigate to EC2 > Key Pairs and click “Import Key Pair.”

   b. Browse to the `nodus.pub` file created above (or paste in the file contents) and use `nodus` as the key pair name.

2.4 AWS Service Limits

AWS accounts come with certain service limits that may restrict the overall bursting experience. They affect things like max number of instances, volumes, Elastic IP's etc. These limits can be found on the AWS Service Limits page. If a limit is found to be too low, visit the Amazon EC2 Service Limits page to find instructions on requesting a limit increase.

2.5 Moab/NODUS Cloud Bursting JSON Scripts

2.5.1 NODUS_Policies_Bursting.JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "VisualEditor0",
            "Effect": "Allow",
            "Action": [
                "ec2:RevokeSecurityGroupIngress",
                "ec2:AuthorizeSecurityGroupEgress",
                "ec2:AuthorizeSecurityGroupIngress",
                "ec2:RevokeSecurityGroupEgress"
            ],
            "Resource": "arn:aws:ec2:*:*:security-group/*"
        },
        {
            "Sid": "VisualEditor1",
            "Effect": "Allow",
            "Action": [
                "ec2:DescribeInstances",
                "ec2:DescribeTags",
                "ec2:CreateKeyPair",
                "ec2:DescribeInstanceAttribute",
                "ec2:RegisterImage",
                "ec2:CreateImage",
                "ec2:DescribeInstanceCreditSpecifications",
                "ec2:DescribeSecurityGroups",
                "ec2:DescribeImages",
                "ec2:ModifyInstanceCreditSpecification",
                "ec2:DescribeVpcs",
                "ec2:CreateSecurityGroup",
                "ec2:DescribeVolumes",
                "ec2:ModifyInstanceAttribute",
                "ec2:DescribeSubnets",
                "ec2:DescribeInstanceStatus"
            ],
            "Resource": "*"
        },
        {
            "Sid": "VisualEditor2",
            "Effect": "Allow",
            "Action": [
                "ec2:AttachVolume",
                "ec2:CreateVolume"
            ],
            "Resource": [
                "arn:aws:ec2:*:*:instance/*",
                "arn:aws:ec2:*:*:volume/*"
            ]
        },
        {
            "Sid": "VisualEditor3",
            "Effect": "Allow",
            "Action": [
                "ec2:TerminateInstances",
                "ec2:StartInstances",
```

```
                "ec2:RunInstances",
                "ec2:StopInstances"
            ],
            "Resource": [
                "arn:aws:ec2:*:*:subnet/*",
                "arn:aws:ec2:*:*:key-pair/*",
                "arn:aws:ec2:*:*:instance/*",
                "arn:aws:ec2:*:*:launch-template/*",
                "arn:aws:ec2:*::snapshot/*",
                "arn:aws:ec2:*:*:volume/*",
                "arn:aws:ec2:*:*:security-group/*",
                "arn:aws:ec2:*:*:placement-group/*",
                "arn:aws:ec2:*:*:network-interface/*",
                "arn:aws:ec2:*::image/*"
            ]
        },
        {
            "Sid": "VisualEditor4",
            "Effect": "Allow",
            "Action": "ec2:CreateTags",
            "Resource": "arn:aws:ec2:*:*:instance/*"
        }
    ]
}
```

2.5.2 NODUS_Policies_Stack_Build.JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "VisualEditor0",
            "Effect": "Allow",
            "Action": [
                "ec2:TerminateInstances",
                "ec2:StartInstances",
                "ec2:CreateTags",
                "ec2:RunInstances",
                "ec2:StopInstances"
            ],
            "Resource": [
                "arn:aws:ec2:*:*:subnet/*",
                "arn:aws:ec2:*:*:key-pair/*",
                "arn:aws:ec2:*:*:instance/*",
                "arn:aws:ec2:*:*:launch-template/*",
                "arn:aws:ec2:*::snapshot/*",
                "arn:aws:ec2:*:*:volume/*",
                "arn:aws:ec2:*:*:security-group/*",
                "arn:aws:ec2:*:*:placement-group/*",
                "arn:aws:ec2:*:*:network-interface/*",
                "arn:aws:ec2:*::image/*"
            ]
        },
        {
            "Sid": "VisualEditor1",
            "Effect": "Allow",
            "Action": [
                "ec2:DescribeImages",
                "ec2:DescribeInstances",
                "ec2:CreateKeyPair",
```

```
                "ec2:CreateSecurityGroup",
                "ec2:CreateImage",
                "ec2:DescribeKeyPairs",
                "ec2:DeleteKeyPair",
                "ec2:DescribeSecurityGroups",
                "ec2:DescribeInstanceStatus"
            ],
            "Resource": "*"
        },
        {
            "Sid": "VisualEditor2",
            "Effect": "Allow",
            "Action": [
                "ec2:AuthorizeSecurityGroupEgress",
                "ec2:AuthorizeSecurityGroupIngress",
                "ec2:UpdateSecurityGroupRuleDescriptionsEgress",
                "ec2:DeleteSecurityGroup",
                "ec2:UpdateSecurityGroupRuleDescriptionsIngress"
            ],
            "Resource": "arn:aws:ec2:*:*:security-group/*"
        }
    ]
}
```

2.5.3 Rapid Deployment Script (td_rapid_deployment.sh)

```
#!/bin/bash

# td_rapid_deployment.sh: easily setup Test Drive instances in preparation for
bursting
# HOSTS FILE FORMAT:
# <PUBLIC IP> <PRIVATE IP> <HOSTNAME>

HOSTS_FILE=$1
KEY_FILE=$2

if [ -z $KEY_FILE ]; then
  USE_KEY=""
else
  USE_KEY="-i $KEY_FILE"
fi
echo "$USE_KEY"

read -r -d '' ETC_HOSTS_PREFIX << EOM
127.0.0.1   localhost localhost.localdomain localhost4 localhost4.localdomain4
::1         localhost localhost.localdomain localhost6 localhost6.localdomain6

18.219.74.231 app.nodusplatform.com
EOM

HOSTS=`cat $HOSTS_FILE`
PUB_IP=($(echo "$HOSTS" | awk '{print $1}'))
PRIV_IP=($(echo "$HOSTS" | awk '{print $2}'))
HOSTNAME=($(echo "$HOSTS" | awk '{print $3}'))
ETC_HOSTS=`echo "$HOSTS" | awk '{print $2 " " $3}'`
SERVERIP=`echo $HOSTS | grep bursting-sched | awk '{print $1}'`
#echo "$ETC_HOSTS"

for ((i=0;i<${#PUB_IP[@]};++i)); do
  printf "*** Configuring %s %s %s ***\n" "${HOSTNAME[i]}" "${PUB_IP[i]}" "${PRIV_IP
```

```
[i]}"
  echo "Setting hostname"
  ssh -o "StrictHostKeyChecking no" $USE_KEY ec2-user@${PUB_IP[i]} "sudo hostnamectl
set-hostname ${HOSTNAME[i]}"
  echo "Updating /etc/hosts"
  ssh -o "StrictHostKeyChecking no" $USE_KEY ec2-user@${PUB_IP[i]} "sudo su -c 'echo
\"$ETC_HOSTS_PREFIX\" > /etc/hosts'"
  ssh -o "StrictHostKeyChecking no" $USE_KEY ec2-user@${PUB_IP[i]} "sudo su -c 'echo
\"$ETC_HOSTS\" >> /etc/hosts'"
  if [ ${HOSTNAME[i]} == "bursting-sched" ]; then
    echo "Configuring Moab Server: ${HOSTNAME[i]}"
    echo "Copying up moab licenses"
    # legacy license
    #scp -o "StrictHostKeyChecking no" $USE_KEY moab.lic ec2-user@${PUB_IP[i]}:~/
    #ssh -o "StrictHostKeyChecking no" $USE_KEY ec2-user@${PUB_IP[i]} "sudo mv
moab.lic /opt/moab/etc"
    # RLM license
    scp -o "StrictHostKeyChecking no" $USE_KEY moab-rlm.lic ec2-user@${PUB_IP[i]}:~/
    ssh -o "StrictHostKeyChecking no" $USE_KEY ec2-user@${PUB_IP[i]} "sudo mv moab-
rlm.lic /opt/moab/etc"
    scp -o "StrictHostKeyChecking no" $USE_KEY moab-rlm-elastic-tracking.lic ec2-
user@${PUB_IP[i]}:~/
    ssh -o "StrictHostKeyChecking no" $USE_KEY ec2-user@${PUB_IP[i]} "sudo mv moab-
rlm-elastic-tracking.lic /opt/rlm"
    scp -o "StrictHostKeyChecking no" $USE_KEY nodus.pem ec2-user@${PUB_IP[i]}:~/
    scp -o "StrictHostKeyChecking no" $USE_KEY nodus.pub ec2-user@${PUB_IP[i]}:~/
    ssh -o "StrictHostKeyChecking no" $USE_KEY ec2-user@${PUB_IP[i]} "sudo mv
nodus.pem /opt/nodus-cloud-cli"
    ssh -o "StrictHostKeyChecking no" $USE_KEY ec2-user@${PUB_IP[i]} "cat nodus.pub >>
~/.ssh/authorized_keys"
    ssh -o "StrictHostKeyChecking no" $USE_KEY ec2-user@${PUB_IP[i]} "sudo su -c 'echo
\"secret1\" | passwd \"ec2-user\" --stdin'"
    echo "Restarting services"
    ssh -o "StrictHostKeyChecking no" $USE_KEY ec2-user@${PUB_IP[i]} "sudo systemctl
restart rlm"
    ssh -o "StrictHostKeyChecking no" $USE_KEY ec2-user@${PUB_IP[i]} "sudo systemctl
restart trqauthd.service"
    ssh -o "StrictHostKeyChecking no" $USE_KEY ec2-user@${PUB_IP[i]} "sudo systemctl
restart pbs_server.service"
    ssh -o "StrictHostKeyChecking no" $USE_KEY ec2-user@${PUB_IP[i]} "sudo systemctl
restart moab.service"
    ssh -o "StrictHostKeyChecking no" $USE_KEY ec2-user@${PUB_IP[i]} "sudo systemctl
restart postgresql.service"
    ssh -o "StrictHostKeyChecking no" $USE_KEY ec2-user@${PUB_IP[i]} "sudo systemctl
restart acfileman.service"
    ssh -o "StrictHostKeyChecking no" $USE_KEY ec2-user@${PUB_IP[i]} "sudo systemctl
restart mongod.service"
    ssh -o "StrictHostKeyChecking no" $USE_KEY ec2-user@${PUB_IP[i]} "sudo systemctl
restart tomcat.service"
    ssh -o "StrictHostKeyChecking no" $USE_KEY ec2-user@${PUB_IP[i]} "sudo systemctl
restart httpd.service"
  else
    echo "Configuring Compute Node: ${HOSTNAME[i]}"
    echo "Mounting NFS share"
    ssh -o "StrictHostKeyChecking no" $USE_KEY ec2-user@${PUB_IP[i]} "sudo mount -t
nfs bursting-sched:/var/share /mnt/share"
    ssh -o "StrictHostKeyChecking no" $USE_KEY ec2-user@${PUB_IP[i]} "sudo mount -t
nfs bursting-sched:/home /home"
    printf "Adding %s to cluster\n" "${HOSTNAME[i]}"
    ssh -o "StrictHostKeyChecking no" $USE_KEY ec2-user@$SERVERIP "sudo
/usr/local/bin/qmgr -c 'create node ${HOSTNAME[i]}'"
    echo "Restarting pbs_mom"
    ssh -o "StrictHostKeyChecking no" $USE_KEY ec2-user@${PUB_IP[i]} "sudo systemctl
```

```
restart pbs_mom.service"
  fi
done
```

Chapter 3: Installing and Configuring Moab/NODUS Cloud Bursting

In this chapter:

3.1 Install and Configure NODUS-Cloud-CLI

Nodus-Cloud-CLI is the package that contains the Elastic scripts and other items in order for Moab to talk to the NODUS server.

In this topic:

3.1.1 Installation Steps

To install NODUS-Cloud-CLI, perform the following steps (described in more detail below):

1. Install Node.js.
2. Install `nodus-cloud-cli`.
3. Link NODUS-Cloud-CLI with Cloud Providers

3.1.2 Install Node.js

Node.js supports Red Hat® Enterprise Linux® / RHEL, CentOS and Fedora.

Supported Red Hat® Enterprise Linux® Versions

- RHEL 5 (32-bit and 64-bit)
- RHEL 6 (32-bit and 64-bit)
- RHEL 7 (64-bit)

Supported CentOS Versions

- CentOS 5 (32-bit and 64-bit)
- CentOS 6 (32-bit and 64-bit)
- CentOS 7 (64-bit)

Node.js is available from the NodeSource Enterprise Linux and Fedora binary distributions repository. Support for this repository, along with its scripts, can be found on GitHub at `nodesource/distributions`.

Note that the Node.js packages for EL 5 (RHEL5 and CentOS 5) depend on the EPEL repository being available. The setup script will check and provide instructions if it is not installed.

Perform the following steps as root on RHEL, CentOS or Fedora.

1. Run the Node.js repo install script.

```
[root]# curl --silent --location https://rpm.nodesource.com/setup_7.x | bash -
```

2. Install build tools.

```
[root]# yum -y install gcc-c++ make
```

3. Install Node.js.

```
[root]# yum -y install nodejs
```

4. To check that Node.js was installed successfully, have Node.js print its version information to the console.

```
[root]# node --version
v7.8.0
```

3.1.3 Install NODUS-Cloud-CLI

To install the `nodus-cloud-cli`, do the following:

1. Download the `nodus-cloud-cli` release package from the Adaptive Download Center (https://www.adaptivecomputing.com/support/download-center/nodus-cloud-cli/).

2. Extract the `nodus-cloud-cli` tarball.

```
[root]# tar -xzvf nodus-cloud-cli-*.tar.gz -C /opt
```

3. Install the `nodus-run` command.

```
[root]# cd /opt/nodus-cloud-cli
[root]# ./install.sh
```

3.1.4 Link NODUS-Cloud-CLI with Cloud Providers

NODUS supports provisioning to many cloud providers including AWS, Azure, and Google.

3.1.4.A Credentials

Credentials for the various cloud providers are stored locally. By default, the elastic trigger scripts look in the path `/opt/nodus-cloud-cli/cloud_credentials/<cloud_provider>/credentials.json`.

Inside the `cloud_credentials` directory, you can store credentials for multiple cloud providers in different directories. If you use multiple credentials for the same provider, you can tag them by group. By default, all credentials are part of the default group.

3.1.4.B AWS Example

AWS credentials are stored in the `/opt/nodus-cloud-cli/cloud_credentials/aws/credentials.json` file.

```
{
    "access_key": "ABCDEFGHIJKLMNOPQRST",
    "secret_key": "AbcDef/gHiJkLmopxvc3/8AbCDEvpRLJ7ugjK4vle"
}
```

3.1.5 Generating Key Pairs

NODUS needs a `nodus` key pair in order to finalize the created instance. The `nodus` key pair can also be used to connect to the bursted instances.

To create the NODUS key pair, perform the following steps:

1. Using the command *`ssh-keygen`*, create a new key pair, calling it `nodus`:

   ```
   % ssh-keygen -t rsa -C "nodus"
   ```

   When *`ssh-keygen`* asks for the file in which to save the key, save the file in an easily accessible location and name it `nodus`.

2. In every region where bursting is to take place, do the following:

   a. In the AWS console, navigate to EC2 > Key Pairs and click “Import Key Pair.”

   b. Browse to the `nodus.pub` file created above (or paste in the file contents) and use `nodus` as the key pair name.

3. Rename the `nodus` private key to `nodus.pem`. This is only to easily identify the file for further reference in this guide.

3.2 Install and Configure NODUS-DNS-Service (Optional)

The `nodus-dns-service` provides a lightweight DNS service with an HTTP/REST based interface that can by updated dynamically with the IPs and hostnames of instances that are created. It uses ports 53 and 5353, which need to be opened on the server instance hosting the DNS service.

- Port 53 is the standard DNS service port and is used to lookup the address of machines on the network by hostname.
- Port 5353 is the HTTP/REST interface, which is used to add/remove DNS records.

These step are assuming you have already installed `Node.js` and the NODUS commands from the NODUS-Cloud-CLI.

In this topic:

3.2.1 Installing NODUS-DNS-Service

To install NODUS-DNS-Service:

1. Download the nodus-dns-service release package from the Adaptive Download Center (https://www.adaptivecomputing.com/support/download-center/nodus-cloud-cli/).
2. Extract the `nodus-dns-service` tarball.

```
[root]# tar -xzvf  nodus-dns-service-*.tar.gz -C /opt
```

3. Install the service dependencies.

```
[root]# cd /opt/nodus-dns-service
[root]# npm install
```

4. Start the DNS service.

```
[root]# nodus start dns
```

3.2.2 Using the NODUS-DNS-Service

In the bootstrap script for the cloud instance, look up the local IP address and add it to the DNS with the hostname:

```
#!/bin/bash
IPADDRESS=`/sbin/ip addr | grep eth0 | tail -n1 | awk '{print $2}' | cut -f1  -d'/'`
curl http://server:5353/v1/records/{{hostname}} -X POST -d "{\"address\":
\"$IPADDRESS\", \"type\": \"A\", \"ttl\": 60}" -H "Content-Type: application/json"
```

`{{hostname}}` is the hostname of the cloud instance and is populated by NODUS.

As long as the server running the NODUS-DNS-Service is listed as a nameserver in `/etc/resolv.conf`, the cloud instance's IP will resolve to its hostname.

3.3 Building Stacks with NODUS

To launch instances in the cloud, a stack must first be created. The stack consists of the base OS, services, libraries, applications, and any data that is needed so that, once the instance is starts up, it is ready to begin processing jobs. This process is fully customizable by using a stack JSON file. Stacks can be built off of base OS images or off of a prebuilt and configured image snapshot.

In this topic:

3.3.1 Creating Stack Definition JSON File

Stacks are defined in a JSON file describing the provisioning dependencies and tasks. The following JSON files are examples of how to create a stack off of a base OS image and a using prebuilt image (e.g., Amazon Machine Image or AMI).

The base OS stack JSON file installs and configures the Torque client (`pbs_mom`), OpenVPN, OpenLDAP, and NFS on a base OS image.

```
{
   "name": "centos",
   "version": "1",
   "image": {
      "source_ami": "ami-78485818",
      "ssh_username": "centos",
      "region": "us-west-1"
   },
   "files": ["torque/RPMs/moab-torque-client-6.1.2-1.el7.x86_64.rpm",
"torque/RPMs/moab-torque-common-6.1.2-1.el7.x86_64.rpm", "torque/RPMs/moab-torque-mom-
6.1.2-1.el7.x86_64.rpm"],
   "bootstrap": "./centos-7-bootstrap.sh",
   "tasks": [{
         "type": "shell",
         "inline": [
            "mkdir -p ~/install"
         ]
      },
      {
         "type": "file",
```

```
         "source": "./torque/RPMs/moab-torque-client-6.1.2-1.el7.x86_64.rpm",
         "destination": "~/install/"
      },
      {
         "type": "file",
         "source": "./torque/RPMs/moab-torque-common-6.1.2-1.el7.x86_64.rpm",
         "destination": "~/install/"
      },
      {
         "type": "file",
         "source": "./torque/RPMs/moab-torque-mom-6.1.2-1.el7.x86_64.rpm",
         "destination": "~/install/"
      },
      {
         "type": "shell",
         "inline": [
            "cat /etc/redhat-release",
            "cd ~/install",
            "echo Install Prerequisite Packages...",
            "sudo yum -y install autofs unzip moab-torque-common-*.rpm moab-torque-
mom-*.rpm moab-torque-client-*.rpm https://dl.fedoraproject.org/pub/epel/epel-release-
latest-7.noarch.rpm",
            "sudo yum -y install openvpn easy-rsa",
            "sudo -- sh -c 'echo \"\\$pbsserver bursting-sched.ac\" >
/var/spool/torque/mom_priv/config'",
            "sudo -- sh -c 'echo \"\\$logevent  225\" >> /var/spool/torque/mom_
priv/config'",
            "sudo -- sh -c 'echo \"\\$loglevel 7\" >> /var/spool/torque/mom_
priv/config'",
            "echo Configuring NFS...",
            "echo Adding home and mounts to /etc/auto.master...",
            "sudo -- sh -c 'echo \"/home  /etc/home.nfs --timeout=60\" >>
/etc/auto.master'",
            "sudo -- sh -c 'echo \"/mnt   /etc/mnt.nfs\" >> /etc/auto.master'",
            "sudo -- sh -c 'echo \"*    -fstype=nfs,rw,nosuid,soft  bursting-
sched.ac:/home/&\" > /etc/nfs.home'",
            "sudo -- sh -c 'echo \"share  -fstype=nfs,rw  bursting-
sched.ac:/var/share\" > /etc/mnt.nfs'"
         ]
      }
   ]
}
```

The following is an example of a stack using a prebuilt EC2 Amazon Machine Image.

```
{
   "name": "ami-build",
   "version": "1",
   "image": {
      "source_ami": "ami-d25948b2",
      "ssh_username": "ec2-user",
      "region": "us-west-1"
   },
   "bootstrap": "ami-build-bootstrap.sh",
   "tasks": []
}
```

3.3.2 Stack Bootstrap Files

Bootstrap files are scripts that are run at the initial creation of a cloud instance. They are generally used for resetting hostnames, configuring networks, starting VPNs, and mounting file systems. The following is an example of a bootstrap file for CentOS 7.

```
#!/usr/bin/env bash

## Stop the pbs_mom service before reconfiguring network layer (i.e. VPN, /etc/hosts/,
hostname, etc..)
echo 'Stopping pbs_mom service...'
sudo systemctl stop pbs_mom

echo 'Displaying VPN IPAddress...'
IPADDRESS=`/sbin/ip addr | grep tun0 | tail -n1 | awk '{print $2}'`
echo "---- VPN IPAddress: $IPADDRESS ----"

## Update HOSTNAME
echo 'Setting hostname={{hostname}}...'
sudo sh set-hostname.sh '{{hostname}}'
echo '----------------------------------------'
echo "HOSTNAME=$( hostname -f )"
echo '----------------------------------------'

## Add TORQUE Server to /etc/hosts file...
echo "Adding pbs_server entry to /etc/hosts..."
sudo -- sh -c 'echo {{args.torque_server_ipaddress}} {{args.torque_server_hostname}}
>> /etc/hosts'

## **** Update DNS and update /etc/resolve.conf
echo
echo 'Registering node with DNS...'
echo "      hostname: {{hostname}}"
echo " vpn_ipaddress: $IPADDRESS"
echo '----------------------------------------'
curl http://bursting-sched:5353/v1/records/{{hostname}} -X POST -d "{\"address\":
\"$IPADDRESS\", \"type\": \"A\", \"ttl\": 60}" -H "Content-Type: application/json"
echo '----------------------------------------'
echo
echo 'Updating resolve.conf...'
echo '----------------------------------------'

sed -i '3 i\nameserver {{args.torque_server_ipaddress}}' /etc/resolv.conf

# Display resolve.conf
cat /etc/resolv.conf

echo '----------------------------------------'
echo

## Start NFS Server
echo
echo 'Enabling NFS service...'
sudo systemctl enable autofs
echo 'Starting NFS service...'
sudo systemctl start autofs
echo
echo 'NFS Service started successfully'

##
```

```
## UPDATE PBS_MOM LEVEL
echo 'Setting pbs_mom log level to 10...'
sudo -- sh -c 'echo "\$loglevel 10" >> /var/spool/torque/mom_priv/config'

## START PBS_MOM

echo 'Restarting pbs_mom service...'
sudo systemctl start pbs_mom
```

3.3.3 Stack Schema

Field Name	Description
name	The name of the stack. This should be unique across all stack definitions.
version	The user-defined version of the stack. If the stack is an application or service package, then the version of the package is recommended.
image	Contains either the base OS name or name of the prebuilt image as well as the `ssh_username` and region name where the stack will reside.
files	The `files` field contains a list of files that will be uploaded to the build server to be used as part of the provisioning process.
tasks	The `tasks` field contains a list of provisioning tasks that will run when the server instance is provisioned. The following task types are referenced below.

3.3.3.A Task: Ansible Local

The `ansible-local` task will run an playbook in "local" mode on the newly provisioned instance using Playbook and Role files. This means Ansible must be installed on the instance. Playbooks and Roles can be uploaded from your build machine using the `files` property. Ansible is then run on the guest machine in local mode via the `ansible-playbook` command.

```
"type": "ansible-local",
"playbook_file": "local.yml"
```

3.3.3.B Task: File

The file task uploads files to machines built by NODUS. The recommended usage of the file task is to use it to upload files, and then use shell task to move them to the proper place, set permissions, etc.

The file task can upload both single files and complete directories.

```
"type": "file",
"source": "app.tar.gz",
"destination": "/tmp/app.tar.gz"
```

3.3.3.C Task: Shell

The shell task provisions machines uses shell scripts to provision new server instances. Shell provisioning is the easiest way to get software installed and configured on a machine.

```
"type": "shell",
"inline": ["echo foo"]
```

The reference of available configuration options is listed below. The only required element is either `inline` or `script`. Every other option is optional.

Exactly one of the following is required.

Option Name	Type	Description
inline	string array	This is an array of commands to execute. The commands are concatenated by newlines and turned into a single file, so they are all executed within the same context. This allows you to change directories in one command and use something in the directory in the next and so on. Inline scripts are the easiest way to pull off simple tasks within the machine.
script	string	The path to a script to upload and execute in the machine. This path can be absolute or relative. If it is relative, it is relative to the working directory when NODUS is executed.
scripts	string array	An array of scripts to execute. The scripts will be uploaded and executed in the order specified. Each script is executed in isolation, so state such as variables from one script will not carry on to the next.

Optional Parameters

Parameter Name	Type	Description
binary	boolean	If true, specifies that the script(s) are binary files, and NODUS should therefore not convert Windows line endings to Unix line endings (if there are any).Default: FALSE.
environment_vars	string array	An array of key/value pairs to inject prior to the execute_command. The format should be key=value. NODUS injects some environmental variables by default into the environment, as well, which are covered in the section below.

Parameter Name	Type	Description
expect_dis-connect	boolean	Whether to error if the server disconnects us. A disconnect might happen if you restart the ssh server or reboot the host. Default: FALSE.
inline_she-bang	string	The shebang value to use when running commands specified by inline. By default, this is `/bin/sh -e`. If you are not using `inline`, then this configuration has no effect. If you customize this, be sure to include something like the `-e` flag, otherwise individual steps failing will not fail the provisioner.
remote_ folder	string	The folder where the uploaded script will reside on the machine. Default: `/tmp`.
remote_file	string	The filename the uploaded script will have on the machine. Default: `script_ nnn.sh`.
remote_path	string	The full path to the uploaded script will have on the machine. Default `remote_ folder/remote_file`. If set, this option will override both `remote_folder` and `remote_file`.
skip_clean	boolean	If TRUE, specifies that the helper scripts uploaded to the system will not be removed by NODUS. Default: FALSE (clean scripts from the system).
start_retry_ timeout	string	The amount of time to attempt to start the remote process. Default: 5m (5 minutes). This setting exists in order to deal with times when SSH may restart, such as a system reboot. Set this to a higher value if reboots take a longer amount of time.

3.3.3.D Task: Powershell

The PowerShell task runs PowerShell scripts on Windows machines. It assumes that the communicator in use is WinRM.

```
"type": "powershell",
"inline": ["dir c:\\"]
```

The reference of available configuration options is listed below. The only required element is either `inline` or `script`. Every other option is optional.

Exactly one of the following is required.

Option Name	Type	Description
inline	string array	This is an array of commands to execute. The commands are concatenated by newlines and turned into a single file, so they are all executed within the same context. This allows you to change directories in one command and use something in the directory in the next and so on. Inline scripts are the easiest way to pull off simple tasks within the machine.
script	string	The path to a script to upload and execute in the machine. This path can be absolute or relative. If it is relative, it is relative to the working directory when NODUS is executed.
scripts	string array	An array of scripts to execute. The scripts will be uploaded and executed in the order specified. Each script is executed in isolation, so state such as variables from one script will not carry on to the next.

Optional Parameters

Option Name	Type	Description
binary	boolean	If true, specifies that the script(s) are binary files, and NODUS should therefore not convert Windows line endings to Unix line endings (if there are any). Default: FALSE.
elevated_execute_command	string	The command to use to execute the elevated script. By default this is as follows: `powershell -executionpolicy bypass "& { if (Test-Path variable:global:ProgressPreference) {$ProgressPreference='SilentlyContinue'};. {{.Vars}}; &'{{.Path}}'; exit $LastExitCode }"` The value of this is treated as configuration template. There are two available variables: `Path`, which is the path to the script to run, and `Vars`, which is the location of a temp file containing the list of `environment_vars`, if configured.
environment_vars	string array	An array of key/value pairs to inject prior to the execute_command. The format should be key=value. NODUS injects some environmental variables by default into the environment, as well, which are covered in the section below.
execute_command	string	The command to use to execute the script. By default this is as follows: `powershell -executionpolicy bypass "& { if (Test-Path variable:global:ProgressPreference) {$ProgressPreference='SilentlyContinue'};. {{.Vars}}; &'{{.Path}}'; exit $LastExitCode }"` The value of this is treated as configuration template. There are two available variables: `Path`, which is the path to the script to run, and `Vars`, which is the location of a temp file containing the list of `environment_vars`, if configured.

Option Name	Type	Description
elevated_user and elevated_password	string	If specified, the PowerShell script will be run with elevated privileges using the given Windows user.
remote_path	string	The path where the script will be uploaded to in the machine. Default: `c:/Windows/Temp/script.ps1`. This value must be a writable location and any parent directories must already exist.
valid_exit_codes	list of ints	Valid exit codes for the script. Default: 0.

3.3.3.E Task: Chef

The Chef solo task installs and configures software on machines using `chef-solo`. Cookbooks can be uploaded from your local machine to the remote machine or remote paths can be used.

The task will even install Chef onto your machine if it is not already installed, using the official Chef installers provided by Chef Inc.

```
"type": "chef-solo",
"cookbook_paths": ["cookbooks"]
```

The reference of available configuration options is listed below. No configuration is actually required, but at least `run_list` is recommended.

Option Name	Type	Description
chef_environment	string	The name of the `chef_environment` sent to the Chef server. By default this is empty and will not use an environment.
config_template	string	Path to a template that will be used for the Chef configuration file. By default NODUS only sets configuration it needs to match the settings set in the provisioner configuration. If you need to set configurations that the NODUS provisioner does not support, then you should use a custom configuration template. See the dedicated "Chef Configuration" section below for more details.
cookbook_paths	string array	This is an array of paths to `cookbooks` directories on your local filesystem. These will be uploaded to the remote machine in the directory specified by the `staging_directory`. By default, this is empty.

Option Name	Type	Description
data_bags_path	string	The path to the `data_bags` directory on your local filesystem. These will be uploaded to the remote machine in the directory specified by the `staging_directory`. By default, this is empty.
encrypted_data_bag_secret_path	string	The path to the file containing the secret for encrypted data bags. By default, this is empty, so no secret will be available.
environments_path	string	The path to the `environments` directory on your local filesystem. These will be uploaded to the remote machine in the directory specified by the `staging_directory`. By default, this is empty.
execute_command	string	The command used to execute Chef. This has various configuration template variables available. See below for more information.
guest_os_type	string	The target guest OS type, either `unix` or `windows`. Setting this to `windows` will cause the provisioner to use Windows friendly paths and commands. By default, this is `unix`.
install_command	string	The command used to install Chef. This has various configuration template variables available. See below for more information.
json	object	An arbitrary mapping of JSON that will be available as node attributes while running Chef.
prevent_sudo	boolean	By default, the configured commands that are executed to install and run Chef are executed with sudo. If this is true, then the sudo will be omitted. This has no effect when `guest_os_type` is `windows`.
remote_cookbook_paths	string array	A list of paths on the remote machine where cookbooks will already exist. These may exist from a previous provisioner or step. If specified, Chef will be configured to look for cookbooks here. By default, this is empty.
roles_path	string	The path to the `roles` directory on your local filesystem. These will be uploaded to the remote machine in the directory specified by the `staging_directory`. By default, this is empty.
run_list	string array	The run list for Chef. By default this is empty.

Option Name	Type	Description
skip_install	boolean	If true, Chef will not automatically be installed on the machine using the Chef omnibus installers.
staging_directory	string	This is the directory where all the configuration of Chef by NODUS will be placed. This directory does not need to exist but must have proper permissions so that the user that NODUS uses is able to create directories and write into this folder. If the permissions are not correct, use a shell provisioner prior to this to configure it properly.
version	string	The version of Chef to be installed. By default this is empty, which will install the latest version of Chef.

3.3.4 Building the Stack

Once the JSON is complete, the stack can be built using the following command:

```
[root]# nodus-run /opt/nodus-cloud-cli/stack/build.js <path to JSON file>
credentials=/opt/nodus-cloud-cli/cloud_credentials/aws/credentials.json
```

This will launch a "Packer Builder" instance in the region specified by "region" field in the JSON file. Once completed, it will return a stack ID that can be used to reference the stack for bursts in that region.

3.4 Configuring Elastic Computing For Use With NODUS

The main use case of Elastic Computing using NODUS is based on a backlog belonging to a QoS. Once the backlog has exceeded the provided trigger threshold, the trigger fires and calls out to NODUS to order a specified number of instances.

This topic provides examples of backlog-based Elastic Computing and node end scripts. Your scripts will vary based on your system configuration. Please contact your Adaptive Computing account manager for suggestions and options to configure Elastic Computing.

If you are using Elastic Computing with Torque, you cannot have a `mom_hierarchy` file in the `$PBS_HOME/server_priv` directory.

In this topic:

3.4.1 To Configure Elastic Computing

1. If you did not perform a manual installation to install Moab Workload Manager, follow the instructions in "Install Moab Server" in the *Moab 9.1.3 Installation and Configuration Guide*.
2. Enable dynamic nodes in the `moab.cfg` file.

   ```
   SCHEDCFG[] FLAGS=enabledynamicnodes
   ```

   A sample excerpt from a `moab.cfg` file is shown in 3.4.2 Sample moab.cfg File Excerpt - page 36 below.
3. If you want to be able to view node and trigger information, use one of these Moab tools:
   - `mdiag -n -v -xml`
   - `mdiag -T`
   - `checknode -v <node name>`

   See 4.2 Viewing Node and Trigger Information - page 48 for more information.
4. If you want to record dynamic node activity, enable NODEADD and/or NODEREMOVE for RECORDEVENTLIST in the `moag.cfg` file. See 4.1 Dynamic Nodes - page 44 for more information.

5. In the `moab.cfg` file, make these changes for QoS triggers:
   a. Add the elastic trigger: `TType=elastic`

      See 3.6 Elastic Triggers With NODUS - page 38 for more information.

   b. Specify how nodes are requested when the trigger fires, using one of these two options:
      - Specify time and number of nodes.

```
QOSCFG[xyz] REQUESTGEOMETRY=12@4:00:00:00
```

When the elastic trigger fires, request 12 additional nodes for 4 days, 0 hours, 0 minutes, and 0 seconds.

> The `REQUESTGEOMETRY` values shown are just an example.

      - Base node request on highest priority job.

```
QOSCFG[xyz] REQUESTGEOMETRY=PRIORITYJOBSIZE
```

When the elastic trigger fires, request enough nodes to run the highest priority job in the backlog for the amount of walltime specified by the highest priority job.

6. Configure Torque for Elastic Computing.

```
[root]# qmgr -c "set server auto_node_np = True"
```

This automatically configures a node's `np` (number of processors) value based on the `ncpus` value from the status update. Requires full manager privilege to set or alter.

7. Use BACKLOGCOMPLETIONTIME to specify when the elastic trigger fires (adding nodes).

> The trigger that contains the `BACKLOGCOMPLETIONTIME` threshold can only be used when profiling is enabled.

The `BACKLOGCOMPLETIONTIME` is calculated by Moab as follows: (The maximum number of processor seconds in the QoS) divided by (The total number of processors in the system). See "`BACKLOGCOMPLETIONTIME`" in the *Moab Workload Manager Administrator Guide* for more information.

```
NODECFG[DEFAULT] ENABLEPROFILING=TRUE
QOSCFG[xyz] TRIGGER=EType=threshold,AType=exec,Action="/opt/nodus-cloud-
cli/elastic.py -g $REQUESTGEOMETRY -f aws-cloud -p aws -s b8133135-2914-4a4f-9ade-
5d4a3bee7045 -i t2.medium -k
~/nodus.pem",Threshold=BACKLOGCOMPLETIONTIME>1800,RearmTime=05:00
```

In the above example, when the `BACKLOGCOMPLETIONTIME` is more than 1800 seconds, the `QOSCFG` threshold trigger fires and the `/opt/nodus-cloud-cli/elastic.py` script executes. This script calls out to NODUS to order the number of nodes specified by `-g $REQUESTGEOMETRY`, which gets passed in from Moab.

The script also applies node feature specified by `-f`. The instances created will be derived from the stack referenced by the stack ID that is specified with `-s`. The stack ID is created using NODUS (see 3.3 Building Stacks with NODUS - page 23). The instance type is specified with `-i`. The instance names will have a prefix name of whatever is specified with `-p` (in this case, `aws`), followed by a unique generated hash. The `-k` flag specifies the path to the `nodus.pem` that corresponds to the key pair named `nodus` in the cloud providers (see 3.1.5 Generating Key Pairs - page 20).

The `-a` flag lets you specify the access control (e.g., `-a user=ted`) to limit the access to a particular user, group, account, etc. Access control works the same as with reservation ACLs discussed under the "Affinity" heading in "Configuring and Managing Reservations"in section 1.1.2.E Affinity in the *Moab Workload Manager Administrator Guide*.

Once the `BACKLOGCOMPLETIONTIME` threshold is reached, the trigger will begin firing. The administrator can configure the trigger to fire once only or periodically until the node is deleted from Torque by the external service.

8. Determine how the dynamic nodes will be removed. See 4.1 Dynamic Nodes - page 44 for more information.

   Use one or both of these methods:

   - Set the `TTL` when creating the node via the RM. This parameter tells Moab to remove the node when the `TTL` has passed.
   - Add the `NODEIDLEPURGETIME` parameter to `moab.cfg`. To turn off the purging of *individual* dynamic nodes output, specify `noidlepurge` in the `varattr` output of the node using Torque. See `$varattr` in the *Torque Resource Manager Administrator Guide*. Alternately, you can use the `varattr` output from the wiki interface. See `VARATTR` in the *Moab Workload Manager Administrator Guide*.

   You can optionally report a `requestid` on each node in the same group.

Nodes without a `requestid` that hit the configured idle purge time are immediately purged. Whereas, nodes with a `requestid` that hit the configured idle purge time are only purged when all the nodes that have the same `requestid` hit the configured idle purge time.

Configure the node end trigger in `moab.cfg`.

```
NODECFG[DEFAULT] TRIGGER=EType=end,TType=elastic,AType=exec,Action="/opt/nodus-
cloud-cli/elastic.py -r $OID"
```

In this example, the `nodeend.sh` trigger will be called with the name of each node in the `requestid` group.

The node end trigger notifies the external service that this node (along with all the other nodes with the same `requestid`) has met the node idle purge time. The external service may then choose to remove the node from Torque (which in turn removes it from Moab).

9. If you want to set limits on whether bursting is available, specify the limits using the usage policies. You can set these limits at the global partition or QoS level. See 4.3 Usage Policies - page 51.

3.4.2 Sample moab.cfg File Excerpt

```
NODECFG[DEFAULT] ENABLEPROFILING=TRUE
SCHEDCFG[moab] FLAGS=enabledynamicnodes
QOSCFG[xyz] REQUESTGEOMETRY=12@4:00:00:00
QOSCFG[xyz] TRIGGER=EType=threshold,AType=exec,Action="/opt/nodus-cloud-cli/elastic.py
-g $REQUESTGEOMETRY -p aws -s b8133135-2914-4a4f-9ade-5d4a3bee7045 -i t2.medium -k
~/nodus.pem",Threshold=BACKLOGCOMPLETIONTIME>1800,RearmTime=05:00
NODEIDLEPURGETIME 3600
NODECFG[DEFAULT] TRIGGER=EType=end,TType=elastic,AType=exec,Action="/opt/nodus-cloud-
cli/elastic.py -r $OID"
JOBRETRYTIME 00:10:00
```

3.4.3 Troubleshooting

3.4.3.A Manual Bursting

Testing bursting can be done manually using the `/opt/nodus-cloud-cli/elastic.py` script, shown below. This script is what Moab uses to call out to NODUS. The script can also be run manually, especially to test the bursting mechanism.

```
[ec2-user ~]$ sudo su
[root ~]$ cd /opt/nodus-cloud-cli
[root ~]$ ./elastic.py -g 5@3:00:00 -f cloud -p aws -s e4816979-ceb8-4aa9-8601-
ce1a4e371af8 -i t2.micro -k /opt/nodus-cloud-cli/nodus.pem
```

This will launch five AWS instances with a TTL of three hours, using the prebuilt stack ID on a `t2.micro` instance. Once the node comes up, the AWS instances should join the cluster and show up in the results of *`pbsnodes`* and *`mdiag -n`*.

3.5 Considering AWS Instance Types

You need to choose the right AWS instance type to ensure that jobs will run effectively in the cloud. Workload will dictate the instance type needed, but in general, batch jobs can usually run on smaller instance types with lower core count, while parallel jobs need larger instances with multiple cores in order to run effectively. Hardware needs are also a consideration. If possible, it is easiest to choose instance types that most closely resemble on-prem resources, but if no instance type can match on-prem resources or price makes that impractical, you may need to benchmark instance types to find which instance type provides the best performance for the workload, while balancing

cost. A good instance type to start with would be `t2.xlarge`, which has 4 cores and 15 GB of memory. Stronger or weaker instances types can be selected from there, depending upon how well they address the given workload. See the AWS instance types page for more details.

3.6 Elastic Triggers With NODUS

When enabled, the elastic trigger allows the Moab scheduler to call out to NODUS to order up instances from the cloud.

The elastic trigger is added to `moab.cfg` when the `TType` trigger component is set to `elastic`. See "Trigger Components" in the *Moab Workload Manager Administrator Guide* for more information.

When configured and enabled, this trigger:

- Takes the `REQUESTGEOMETRY` parameter and creates nodes in provider.
- Adds nodes to Torque using the create node qmgr command (or optionally add it to their RM's cluster query).
- Makes sure that `TTL` is set correctly on the new nodes.
- Optionally adds a request ID (generated by the script) and/or ACL to the nodes.

Example:

```
NODECFG[DEFAULT] ENABLEPROFILING=TRUE
QOSCFG[xyz] TRIGGER=EType=threshold,AType=exec,Action="/opt/nodus-cloud-cli/elastic.py
-g $REQUESTGEOMETRY -f aws-cloud -p aws -s b8133135-2914-4a4f-9ade-5d4a3bee7045 -i
t2.medium -k ~/nodus.pem",Threshold=BACKLOGCOMPLETIONTIME>1800,RearmTime=05:00
```

In the above example, when the `BACKLOGCOMPLETIONTIME` is more than 1800 seconds, the `QOSCFG` threshold trigger fires and the `/opt/nodus-cloud-cli/elastic.py` script executes. This script calls out to NODUS to order the number of nodes specified by `-g $REQUESTGEOMETRY`, which gets passed in from Moab.

The script also applies node feature specified by `-f`. The instances created will be derived from the stack referenced by the stack ID that is specified with `-s`. The stack ID is created using NODUS (see 3.3 Building Stacks with NODUS - page 23). The instance type is specified with `-i`. The instance names will have a prefix name of whatever is specified with `-p` (in this case, `aws`), followed by a unique generated hash. The `-k` flag specifies the path to the `nodus.pem` that corresponds to the key pair named `nodus` in the cloud providers (see 3.1.5 Generating Key Pairs - page 20).

The `-a` flag lets you specify the access control (e.g., `-a user=ted`) to limit the access to a particular user, group, account, etc. Access control works the same as with reservation ACLs discussed under the "Affinity" heading in "Configuring and Managing Reservations"in section 1.1.2.E Affinity in the *Moab Workload Manager Administrator Guide*.

The `BACKLOGCOMPLETION` time trigger threshold may only be used when profiling is enabled.

3.7 Configuring On-Demand Elastic Computing For Use With NODUS

On-demand Elastic Computing is different from backlog-based Elastic Computing in that it allows jobs to specify that they are meant to run in the cloud upon execution, without regard to the existence or state of a job backlog. Once a job is submitted, resources in the cloud are allocated so that the job can start running immediately on those cloud resources. Jobs can be configured to be submitted for on-demand Elastic Computing by using Moab job templates.

This topic provides examples of on-demand Elastic Computing and node end scripts. Your scripts will vary based on your system configuration. Please contact your Adaptive Computing account manager for suggestions and options to configure Elastic Computing.

If you are using Elastic Computing with Torque, you cannot have a `mom_hierarchy` file in the `$PBS_HOME/server_priv` directory.

In this topic:

3.7.1 To Configure Elastic Computing

1. If you installed Moab Workload Manager from the tarball (Manual Installation), ensure you installed `acpython-base` RPM on the Moab Head Node.

   ```
   [root]# rpm -qa|grep acpython-base
   ```

   If you did not perform a manual installation, follow the instructions in "Install Moab Server" in the *Moab 9.1.3 Installation and Configuration Guide*.

2. Enable dynamic nodes in the `moab.cfg` file.

   ```
   SCHEDCFG[] FLAGS=enabledynamicnodes
   ```

   A sample excerpt from a `moab.cfg` file is shown below.

3. If you want to be able to view node and trigger information, use one of these Moab tools:

   - `mdiag -n -v -xml`
   - `mdiag -T`

- `checknode -v <node name>`

See 4.2 Viewing Node and Trigger Information - page 48 for more information.

4. If you want to record dynamic node activity, enable `NODEADD` and/or `NODEREMOVE` for `RECORDEVENTLIST` in the `moag.cfg` file. See 4.1 Dynamic Nodes - page 44 for more information.
5. In the `moab.cfg` file, create job templates for each type of instance, NODUS `stack_id`, and number of instances:

```
JOBCFG[aws-t2-micro] TRIGGER=EType=create,AType=exec,Action="/opt/nodus-cloud-
cli/elastic-ondemand.py -j $JOBID -o $OWNER -s <STACK_ID> -g <REQUESTGEOMETRY> -k
<KEYFILE>",timeout=5:00
```

You must provide the stack ID, request geometry, and the private key file. NODUS generates the stack ID when building the stack in the cloud. You provide the request geometry with the `-g` flag, in the format `num_instances@TTL`. For example, a request geometry of `4@1:00:00:00` specifies 4 instances for one day. Use `-k` to specify the private key file. You can create multiple job templates for different stack IDs and different request geometries.

For more information on job templates, their configuration, and using them to match job attributes, see "About Job Templates" in the *Moab Workload Manager Administrator Guide*.

6. Configure Torque for Elastic Computing.

```
[root]# qmgr -c "set server auto_node_np = True"
```

This automatically configures a node's `np` (number of processors) value based on the `ncpus` value from the status update. Requires full manager privilege to set or alter.

7. Determine how the dynamic nodes will be removed. See 4.1 Dynamic Nodes - page 44 for more information.

Use one or both of these methods:

- Set the `TTL` when creating the node via the RM. This parameter tells Moab to remove the node when the `TTL` has passed.
- Add the `NODEIDLEPURGETIME` parameter to `moab.cfg`. To turn off the purging of *individual* dynamic nodes output, specify `noidlepurge` in the `varattr` output of the node using Torque. See `$varattr` in the *Torque Resource ManagerAdministrator Guide*. Alternately, you can use the `varattr` output from the wiki interface. See `VARATTR` in the *Moab Workload ManagerAdministrator Guide*.

You can optionally report a `requestid` on each node in the same group.

Nodes without a `requestid` that hit the configured idle purge time are immediately purged. Whereas, nodes with a `requestid` that hit the configured idle purge time are only purged when all the nodes that have the same `requestid` hit the configured idle purge time.

Configure the node end trigger in `moab.cfg`.

```
NODECFG[DEFAULT] TRIGGER=EType=end,TType=elastic,AType=exec,Action="/opt/nodus-
cloud-cli/elastic.py -r $OID"
```

In this example, the `nodeend.sh` trigger will be called with the name of each node in the `requestid` group.

The node end trigger notifies the external service that this node (along with all the other nodes with the same `requestid`) has met the node idle purge time. The external service may then choose to remove the node from Torque (which in turn removes it from Moab).

8. If you want to set limits on whether bursting is available, specify the limits using the usage policies. You can set these limits at the global partition or QoS level. See 4.3 Usage Policies - page 51.

3.7.2 Sample moab.cfg File Excerpt

```
SCHEDCFG[moab] FLAGS=enabledynamicnodes
# job template to run jobs in cloud on-demand
JOBCFG[aws-t2-micro] FLAGS=aws-cloud SELECT=TRUE
JOBCFG[aws-t2-micro] TRIGGER=EType=create,AType=exec,Action="/opt/moab/contrib/nodus-
cloud-cli/elastic-ondemand.py -j $JOBID -o $OWNER -s <STACK_ID> -g <REQUESTGEOMETRY> -
k <KEYFILE>",timeout=5:00
NODEIDLEPURGETIME 3600
NODECFG[DEFAULT] TRIGGER=EType=end,TType=elastic,AType=exec,Action="/opt/nodus-cloud-
cli/elastic.py -r $OID"
```

3.7.3 On-Demand Job Submission

In order for jobs to wait for cloud resources to become available, the job must be submitted with a hold in place. Moab will then accept the job submission but not try to schedule it. To submit a job with a hold in place, include the `-h` flag in the `msub` command. Once the cloud instances are up and have registered with `pbs_server` and Moab, the `elastic-ondemand.py` trigger script creates a reservation on the nodes, reserving it for the job, and then it releases the hold on the job automatically so it can begin execution.

```
[root]# msub -l template=aws-t2-micro -h my_job.sh
```

This job is submitted with a hold in place and requests the `aws-t2-micro` job template.

3.7.4 Tuning Backlog Bursting

Backlog bursting happens when the `BacklogCompletionTime` in the threshold trigger is reached. You can see the `BacklogCompletionTime` using *`mdiag -T -v`*:

```
[root@bursting-sched nodus-cloud-cli]# mdiag -T -v
TrigID                Object ID               Event    TType  AType
ActionDate       State
--------------------- --------------------- -------- -------- ------ ------------------
```

```
-- -----------
2                    qos:elastic          threshol  elastic   exec
-     Blocked
Flags:          multifire,globaltrig
BlockUntil:     INFINITY  ActiveTime:  ---
Timeout:        00:05:00
Threshold:      BacklogCompletionTime > 30.00  (current value: 0.00)
Trigger Type:   elastic
RearmTime:      00:15:00
Action Data:    /opt/nodus-cloud-cli/elastic.py -g $REQUESTGEOMETRY -f cloud -p aws -s
b7b8b868-7fef-4c08-80fe-a2e30a19151f -i t2.micro -k /opt/nodus-cloud-cli/nodus.pem
NOTE:  trigger cannot launch - threshold not satisfied - requires usage 0.000000 >
30.000000
4                    node:DEFAULT              end  elastic   exec
-     Blocked
Flags:          globaltrig
BlockUntil:     INFINITY  ActiveTime:  ---
Trigger Type:   elastic
Action Data:    /opt/nodus-cloud-cli/elastic.py -r $OID
```

As jobs queue up in the idle queue, use the `BacklogCompletionTime` in the *`mdiag -T -v`* output to determine the threshold setting for a burst to occur.

Chapter 4: Additional Information

In this chapter:

4.1 Dynamic Nodes

Dynamic nodes are nodes that can be added and removed from Torque at any time. Specifically, any node that has a TTL (time to live) is considered a dynamic node. The following section explains how to add and delete nodes via `qmgr`.

As of Moab version 9.1.2, dynamic node procs are no longer counted against the total procs listed in the Moab license. This allows you to do as many bursts as you desire without exceeding the total procs used for on-premises nodes. If your version of Moab is before 9.1.2, please contact your Adaptive Computing sales representative.

In this topic:

4.1.1 Dynamic Node Parameters

The table below describes the parameters that are used while adding and removing dynamic nodes.

Parameter Name	Required/ Optional	Data Format	Description
TTL	Optional	yyyy-mm-ddThh:mm:ss±hh OR yyyy-mm-ddThh:mm:ss±hhmm OR yyyy-mm-ddThh:mm:ssZ	Time, given as a UTC time, for the node to be removed. The time is Greenwich Mean Time with either an offset or a Z to indicate zero offset.

Parameter Name	Required/ Optional	Data Format	Description
requestid	Optional	Any sequence of non-white-space characters	Identifier used by Moab to identify a group of nodes. See requestid Parameter (Adding or Removing Nodes) for more information.
acl	optional	user==user1:user2,host==host1	List of credentials that can run jobs on this dynamic node.

4.1.2 Dynamic Node Events

You can record dynamic node activity using RECORDEVENTLIST in the `moab.cfg` using one or both of these events:

- NODEADD
- NODEREMOVE

4.1.2.A NODEADD

The `NODEADD` event is generated when the RM first reports a new node to Moab.

The following is an example from the `event_xxx` file in the `$MOAB_HOME/stats` directory:

```
16:22:32 1412202152:359437 node     nuc2         NODEADD      nuc2 STATE=Idle
PARTITION=bdaw ADISK=1 AMEMORY=15193 APROC=4 ASWAP=16717 CDISK=1 CMEMORY=15918 CPROC=4
CSWAP=17442 OS=linux RM=bdaw NODEACCESSPOLICY=SHARED CCLASS=[DevQ][batch] MSG='Node
'nuc2' was newly reported in the last cluster query.  RequestID = 1234, TTL =
1420070400'
```

4.1.2.B NODEREMOVE

The `NODEREMOVE` event is generated when Moab removes a dynamic node after `TTL` has expired, or if the node is no longer reported to Moab by the RM.

The following is an example from the `event_xxx` file in the `$MOAB_HOME/stats` directory:

```
16:21:44 1412202104:359401 node     nuc2         NODEREMOVE   nuc2 STATE=Idle
PARTITION=bdaw ADISK=1 AMEMORY=15192 APROC=4 ASWAP=16716 CDISK=1 CMEMORY=15918 CPROC=4
CSWAP=17442 OS=linux RM=bdaw NODEACCESSPOLICY=SHARED FEATURE=[DEV] CCLASS=[DevQ]
[batch] MSG='Dynamic node 'nuc2' is being removed.  RequestID = 1234, TTL =
1420070400, Reason = node removed because the RM did not report it in the cluster
query'
```

4.1.3 Configuring Dynamic Nodes

This section contains information on configuration options when adding or removing nodes.

- TTL Parameter (Creating Nodes)
- requestid Parameter (Adding or Removing Nodes)
- NODEIDLEPURGETIME Parameter (Removing Nodes)

During the creation of a dynamic node, the pbs_server will attempt to resolve the node name to an IP address. If pbs_server is unable to resolve the name, it will not create the node; nor will it retry the creation later.

Immediately after a dynamic node is created, it is assigned a state of "down|MOM-list-not-sent". Once the new node has received the list of all moms, it will be assigned a state of "free" and be available for job scheduling.

4.1.3.A TTL Parameter (Creating Nodes)

The dynamic nodes are added to the RM with a `TTL` parameter. The `TTL` parameter is passed to Moab by the RM. Moab does not schedule workload for a node beyond the `TTL` assigned to it. Moab removes a dynamic node when it reaches its expiration date as set by `TTL`. A node end trigger will then fire to notify the service that the dynamic node has been removed in Moab and the service may destroy the virtual machine or deprovision the physical nodes at its convenience.

The following is an example of a node being created with a `TTL` parameter:

```
qmgr -c 'create node node003[,node004,node005...] [np=n,][TTL=2015-05-16T05:26:30Z,]
[acl="user==user1:user2:user3",][requestid=n]'
```

In the above example, `node003` is created with `TTL=2015-05-16T05:26:30Z` as the `TTL` parameter. The dynamic node will be removed when the `TTL` is expired.

4.1.3.B requestid Parameter (Adding or Removing Nodes)

The dynamic nodes are added to the RM with a `requestid` parameter that is passed to Moab by the RM. Moab reports the `requestid` parameter along with the node ID in Moab logs, events, and node end triggers. This allows the external service to tag the nodes allocated together in a block. The tagged nodes are then associated as events, and are reported on a node-by-node basis by Moab.

The `requestid` can also be used by the external service to de-allocate nodes together in the same block as they were created by the service. For example, a group of nodes has their node end trigger fired due to node idle purge time or `TTL` expiration.

The requestid is useful if nodes are dynamically added, removed, and then re-added at some later time with the same node ID. Using a requestid when a node is re-added, will help identify each unique instance of a dynamic node's lifetime in logs, events, etc.

Moab also uses the requestid with the NODEIDLEPURGETIME parameter. The requestid parameter groups the nodes and then references the NODEIDLEPURGETIME information, if specified, to determine when to remove the group of nodes. When all the nodes associated with the requestid have reached the idle purge time threshold defined by the NODEIDLEPURGETIME parameter, Moab fires the node end trigger for all the nodes with the same requestid.

When requestid is configured with NODEIDLEPURGETIME, *all* of the nodes must be idle.

4.1.3.C NODEIDLEPURGETIME Parameter (Removing Nodes)

The NODEIDLEPURGETIME parameter instructs Moab to fire a node end trigger when all the nodes in the requestid group have been idle for the time period specified by NODEIDLEPURGETIME.

Setting the NODEIDLEPURGETIME to 0 effectively disables the NODEIDLEPURGETIME. The default value is 0 if NODEIDLEPURGETIME is not configured in the moab.cfg file. See "NODEIDLEPURGETIME" in the *Moab Workload Manager Administrator Guide* for more information.

The following is an example of configuring the node end trigger in moab.cfg

```
NODECFG[DEFAULT]
TRIGGER=EType=end,TType=elastic,AType=exec,Action="/$HOME/tools/nodeend.sh $OID"
```

In this example, the nodeend.sh trigger will be called with the name of each node in the requestid group.

The node end trigger notifies the external service that the node (along with all the other nodes with the same requestid) has met the node idle purge time set by the NODEIDLEPURGETIME parameter. The external service may then choose to remove the node from Torque (which in turn removes it from Moab).

The following is an example of the command that a service will run to remove a node from Torque.

```
qmgr -c 'delete node node003'
```

If a job is running on a node when it is deleted, the job will be requeued if the job is requeueable or deleted if it is not. If the node has already been shut down, any jobs running on the node will be immediately purged.

4.2 Viewing Node and Trigger Information

You can optionally configure Elastic Computing to allow you to view the node and trigger information using the Moab commands described in this topic.

In this topic:

4.2.1 mdiag -n -v --xml

The *mdiag -n -v --xml* command provides detailed information about the state of nodes that Moab is currently tracking. See `mdiag -n` in the *Moab Workload Manager Administrator Guide* for more information.

In the following example, the *mdiag -n - v --xml* command shows the current list of nodes including dynamic node parameters `TTL` and `REQUESTID` in the XML format.

```
$ mdiag -n -v --xml | xmllint --format -
<?xml version="1.0"?>
<Data>
  <node ACL="USER=%=bdaw+:%=adaptive+;" AVLCLASS="[DevQ][batch]" CFGCLASS="[DevQ]
[batch]" FEATURES="DEV" LASTUPDATETIME="1412200545" LOAD="0.330000" MAXJOB="0"
MAXJOBPERUSER="0" MAXLOAD="0.000000" NODEID="bdaw" NODEINDEX="0" NODESTATE="Idle"
OS="linux" OSLIST="linux" PARTITION="bdaw" PRIORITY="0" PROCSPEED="0" RADISK="1"
RAMEM="9746" RAPROC="1" RASWAP="26128" RCDISK="1" RCMEM="16050" RCPROC="1"
RCSWAP="32432" REQUESTID="1234" RESCOUNT="1" RMACCESSLIST="bdaw" RSVLIST="bdaw-TTL-
1234" SPEED="1.000000" STATACTIVETIME="2109" STATMODIFYTIME="1412181806"
STATTOTALTIME="2164684" STATUPTIME="2164668" TTL="1441778400" VARATTR="DEV"/>
  <node AVLCLASS="[DevQ][batch]" CFGCLASS="[DevQ][batch]" CPUCLOCK="OnDemand:800mhz"
FEATURES="DEV" LASTUPDATETIME="1412200545" MAXJOB="0" MAXJOBPERUSER="0"
MAXLOAD="0.000000" NODEID="nuc2" NODEINDEX="2" NODESTATE="Idle" OS="linux"
OSLIST="linux" PARTITION="bdaw" PRIORITY="0" PROCSPEED="0" RADISK="1" RAMEM="15193"
RAPROC="4" RASWAP="16717" RCDISK="1" RCMEM="15918" RCPROC="4" RCSWAP="17442"
RMACCESSLIST="bdaw" SPEED="1.000000" STATACTIVETIME="34" STATMODIFYTIME="1412114379"
STATTOTALTIME="86507" STATUPTIME="86475" VARATTR="DEV"/>
</Data>
```

4.2.2 mdiag -T

The *mdiag -T* command is used to display information about each trigger. See `mdiag -T` in the *Moab Workload Manager Administrator Guide* for more information.

In the following example, the current list of triggers is displayed using the *mdiag -T* command. Notice the node end triggers associated with nodes.

```
$ mdiag -T
```

```
TrigID                Object ID               Event  AType           ActionDate
State
--------------------- ------------------- -------- ------ -------------------- ------
-----
83                    node:DEFAULT              end   exec                    -
Blocked
85*                   node:nuc2                 end   exec          -1:00:01:10
Successful
84*                   node:bdaw                 end   exec             -5:17:23
Active
* indicates trigger has completed
```

4.2.3 checknode -v <node name>

The `checknode -v <node name>` command shows detailed state information and statistics including the `TTL`, the `access control list (ACL)` and the `requestid` for nodes that run jobs. See `checknode` in the *Moab Workload Manager Administrator Guide* for more information.

In the following example, a reservation is created on the node at the `TTL` so that the jobs are not scheduled on the node beyond the `TTL`. Also, a node end trigger is configured on this node which will fire when the node is removed.

```
$ checknode -v bdaw
node bdaw

State:      Idle  (in current state for 5:18:38)
Configured Resources: PROCS: 1  MEM: 15G  SWAP: 31G  DISK: 1M
Utilized   Resources: MEM: 6230M  SWAP: 6230M
Dedicated  Resources: ---
Attributes:         DEV
ACL:        USER==bdaw+:==adaptive+
  MTBF(longterm): 1:00:31:02  MTBF(24h):   INFINITY
Opsys:      linux     Arch:      ---
Speed:      1.00      CPULoad:   0.340
Partition:  bdaw  Rack/Slot:  ---
Features:   DEV
IdleTime:   23:38:11
Classes:    [DevQ][batch]
RM[bdaw]*:  TYPE=PBS
EffNodeAccessPolicy: SHARED
RequestID:  1234
TTL:  Wed Sep  9 00:00:00 2015


Total Time: 25:01:24:09  Up: 25:01:23:53 (100.00%)  Active: 00:35:09 (0.10%)

Reservations:
  bdaw-TTL-1234x1  User     342days ->   INFINITY (  INFINITY)
    Blocked Resources@   342days  Procs: 1/1 (100.00%)  Mem: 16050/16050 (100.00%)
Swap: 32432/32432 (100.00%)  Disk: 1/1 (100.00%)
TrigID                Object ID               Event  AType           ActionDate
State
--------------------- ------------------- -------- ------ -------------------- ------
-----
84*                   node:bdaw                 end   exec  Wed Oct  1 10:43:26
Active
```

```
   Launch Time:   -00:00:14
   Flags:         globaltrig
   Last Execution State: Active (ExitCode: 0)
   BlockUntil:    5:18:24  ActiveTime:  -1:00:29:57
   PID:           7088
   Action Data:   /home/bdaw/nodeend.sh $OID
   StdOut:        /opt/moab/spool/nodeend.sh.oMNnWkU
   StdErr:        /opt/moab/spool/nodeend.sh.ennUbAp

* indicates trigger has completed
```

4.3 Usage Policies

As part of your Elastic Computing solution, you can keep track of processor seconds on all dynamic nodes to limit over-bursting. For example, if your configuration allows 1000 processor seconds of use every day, then if a job needs to burst (and the used processor seconds reaches 1000 before the job can burst), the trigger to burst the job will not fire, and an error message is generated. You can view the error message using "mdiag -T -v".

In this topic:

4.3.1 Available Policies

There are four different values you can set: day, month, quarter, or year. The second count resets at the beginning of each period. For ease of use, you can choose to set the limits based on processor hours, and the system will automatically convert the hours to seconds.

These are the available policies you can set in the moab.cfg file to limit over-bursting:

- To specify by processor seconds, use:
  - MAXDAILYELASTICPROCSECONDS
  - MAXMONTHLYELASTICPROCSECONDS
  - MAXQUARTERLYELASTICPROCSECONDS
  - MAXYEARLYELASTICPROCSECONDS
- To specify by processor hours, use:
  - MAXDAILYELASTICPROCHOURS
  - MAXMONTHLYELASTICPROCHOURS
  - MAXQUARTERLYELASTICPROCHOURS
  - MAXYEARLYELASTICPROCHOURS

4.3.2 Policy Levels

You can set the usage policies at the global partition or QoS level.

- Global Partition – Once the elastic node first appears, Moab will begin keeping track of its processor seconds or hours. If the processor seconds reaches the limit, it will not fire off the elastic trigger so no new nodes will come in. For example:

```
PARCFG[ALL]             MAXDAILYELASTICPROCSECONDS=1000
```

You can view the used and remaining limits using "showstats -v".

- QoS – Processor seconds or hours start being counted once a job is submitting using that particular QOS, not from when the node first appears. For example:

```
QOSCFG[HIGH]            MAXDAILYELASTICPROCSECONDS=500
```

A job is submitted requesting the "HIGH" QOS; the processor seconds begin ticking up for that QOS.

You can view the used and remaining limits using "mdiag -q -v".

4.4 Dealing With Instance Failure

When AWS instances go down, or when networking problems occur, making instances unreachable, Moab is capable of handling these situations based on its Node Availability Policies. Different actions can be selected when this occurs using the `JOBACTIONONNODEFAILURE` option that provides the ability to cancel, fail, hold, ignore, notify, or requeue the job.

See Node Availability Policies in the *Moab Workload Manager Administrator Guide* for more information.

4.5 Cloud Recovery

Because stack building in AWS is automated and can be done from on-prem environments, it makes it easy to recover from any AWS failures that might produce a complete loss of data. Should this occur, NODUS can rebuild the stacks in the cloud in a timely manner (less than 2 hours per stack generated) and, after recreating the AWS config (as discussed Chapter 2: Configuring AWS for Cloud Bursting - page 7), the new AWS account can be up and running, ready for bursting.

4.5.1 Conducting Recovery Testing

If conducting a recovery test is desired, perform the following steps:

1. Create a new AWS account.
2. Follow the steps in Chapter 2: Configuring AWS for Cloud Bursting - page 7 to set up the AWS account for bursting.
3. Update the access and secrets keys mentioned in 3.1.4.A Credentials - page 20 used in the new AWS account.
4. Follow the steps in 3.3 Building Stacks with NODUS - page 23 to rebuild the stack using the JSON file and other config files that were used to create the stack initially.
5. Test bursting following the manual bursting steps in 3.4 Configuring Elastic Computing For Use With NODUS - page 33.

4.6 License Management

Software license management in Moab is typically enabled in one of two models: node-locked and floating. Because cloud nodes are ephemeral, floating licenses are used to allow licenses to be applied to jobs running on those cloud nodes. In a floating license model, a limited number of software licenses are made available cluster wide, and these licenses may be used on any combination of compute hosts. These licenses are consumable and application access is denied once they are gone.

Moab monitors license usage and only launches an application when required software license availability is guaranteed. In addition, Moab also reserves licenses in conjunction with future jobs to ensure these jobs can run at the appropriate time.

For more information on license management, see License Management in the Moab Workload Manager Administrator Guide.