

Nitro 2.0.1

Administrator Guide

August 2016



© 2016 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.
1712 S. East Bay Blvd., Suite 300
Provo, UT 84606
+1 (801) 717-3700
www.adaptivecomputing.com



Scan to open online help

Welcome	1
Chapter 1 Nitro Overview	3
Nitro Origins And Purpose	3
Workload Solutions And Use Cases	4
Theory Of Operation	6
Nitro And System Scheduler Policies	10
Key Terminology And Usage	11
Chapter 2 Installation And Configuration	13
Understand And Plan Your System Environment	13
System Requirements	14
Manual Installation And Upgrade	15
Preparing For Manual Installation Or Upgrade	16
Installing	17
Installing RLM Server	17
Installing Nitro	20
Installing Nitro Web Services	24
Upgrading	32
Upgrading RLM Server	32
Upgrading Nitro	33
Upgrading Nitro Web Services	35
RPM Installation And Upgrade	36
Preparing For RPM Installation Or Upgrade	37
Installing	39
Installing RLM Server	39
Installing Nitro	41
Installing Nitro Web Services	45
Upgrading	53
Upgrading RLM Server (RPM)	53
Upgrading Nitro	53
Upgrading Nitro Web Services (RPM)	55
Chapter 3 System Administration	57
Scheduler And Resource Manager Integration	57
File System Configuration	58
Run Nitro Without A Scheduler	59
Chapter 4 Using Nitro	63
Prepare A Nitro Job	63
Submit A Nitro Job To A Scheduler	67
Track Job Progress	70
Dynamic Workload	76

Chapter 5 References	79
Nitro Configuration File	79
Resource Manager Launch Scripts	80
Command Line Flags, Options, And Positional Parameters	82
Task File	86
Nitrostat	89
Job Recovery	91
Coordinator Resiliency	92
Glossary	92
Chapter 6 Troubleshooting	93
Sources Of Troubleshooting Information	93
Troubleshooting Task Errors	93

Welcome

Welcome to the Administrator Guide for Nitro 2.0.1.

The following chapters are provided to assist in understanding, getting started with, and using Nitro.

- [Nitro Overview on page 3](#) - Provides basic information on Nitro, including theory of operation.
- [Installation and Configuration on page 13](#) - Provides basic installation, configuration, and upgrade information.
- [System Administration on page 57](#) - Contains procedures and reference information for system administrators.
- [Using Nitro on page 63](#) - Contains procedures and reference information on using Nitro.
- [References on page 79](#) - Provides additional conceptual information about Nitro, including a glossary of key terms used throughout this guide.
- [Troubleshooting on page 93](#) - Identifies common sources of reference for troubleshooting and provides troubleshooting information for task errors.

Chapter 1 Nitro Overview

Nitro is the Adaptive Computing High Throughput Computing (HTC) product designed to integrate with either High Performance Computing (HPC), such as Moab Workload Manager, or datacenter schedulers to schedule and run workloads consisting of large quantities (tens of thousands to millions) of small jobs (seconds to minutes to complete) without affecting the throughput of the HPC or datacenter scheduler.

In this chapter:

- [Nitro Origins and Purpose on page 3](#)
- [Workload Solutions and Use Cases on page 4](#)
- [Theory of Operation on page 6](#)
- [Nitro and System Scheduler Policies on page 10](#)
- [Key Terminology and Usage on page 11](#)

Nitro Origins and Purpose

This topic describes the motivation and purpose behind the creation of Nitro.

In this topic:

- [Origins on page 3](#)
- [Purpose on page 4](#)

Origins

Until recently, Adaptive Computing's traditional market has been scheduling batch jobs for HPC systems, colloquially known as "supercomputers". These HPC systems are typically composed of hundreds to tens of thousands of "compute nodes" (servers designed for fast computations and large amounts of I/O), which are increasingly "off-the-shelf" servers. Part of scheduling workloads on such large HPC systems is optimizing the use of all the system's resources. This can be a complex process involving a large amount of computation which can take a significant amount of time.

Complicating the scheduling of HPC systems is the increasing use of workloads consisting of small jobs. These workloads do not need multiple servers but can execute on a single server or even on a single core, and may execute in a short amount of time, sometimes just seconds or even sub-seconds. Such workloads typically fall under the category of HTC and do not need the optimization performed by HPC schedulers.

When an HPC scheduler must schedule large quantities of HTC workloads, these workloads severely slow down the HPC scheduler and reduce its scheduling throughput. To counteract the effects of HTC workloads on HPC schedulers, Adaptive Computing created Nitro. Nitro schedules and runs HTC workloads on any HPC cluster or in any datacenter without affecting the throughput of the HPC or the datacenter scheduling software.

Nitro runs large quantities of HTC workloads as a single job submitted to any HPC or datacenter scheduler (it is scheduler-agnostic) using the resources allocated to it by the HPC or datacenter

scheduler. This permits the scheduler to operate normally and eliminates the impact the scheduling of thousands or millions of HTC workloads would have on it.

Purpose

Nitro runs workloads small enough to execute on a single compute node or server, regardless whether HTC in nature, with as little overhead as possible in order to speed up the execution of HTC-like workloads through the elimination of regular scheduler overhead. This is its primary intended purpose.

Nitro runs a single workload submitted to any HPC or datacenter scheduler and executes hundreds, thousands, or even millions of HTC-like workloads using the resources allocated to it by the HPC or datacenter scheduler. This permits the scheduler to operate normally by eliminating the impact that scheduling of thousands or millions of HTC workloads would have on the scheduler.

Workload Solutions and Use Cases

This topic provides an operational overview of workload solutions available with Nitro. Use cases are also provided to showcase some of the solutions and benefits with using Nitro.

In this topic:

- [Workloads on page 4](#)
- [Use Cases on page 4](#)

Workloads

Nitro easily schedules and executes these typical workloads:

- Serial applications that use only a single core.
- Multi-threaded applications that run on a single host.
- Short-running applications.
- "Embarrassingly Parallel" applications such as Monte Carlo-based simulations.
- Serial or parallel applications that run on a single host.
- Regression testing.

Use Cases

This section contains use cases of Nitro workload solutions.

In this section:

- [Many Independent Short Workloads on page 5](#)
- [Large Queues on page 5](#)
- [Multi-threaded HTC Applications on page 5](#)
- [Regression Testing on page 6](#)

Many Independent Short Workloads

Let's say a user wants to submit a workload of 50,000 HTC jobs to execute on a system, this means submitting each job separately to the system's scheduler. Submitting all 50,000 jobs at once to the work queue slows down the scheduler sufficiently that other users complain of reduced job response times (turnaround time between job submission and job completion). This reduced response time is due to the overhead the scheduler incurs scheduling so many HTC jobs and the overhead of starting and managing so many individual short jobs. In other words, the shorter the HTC job, the greater the percentage of the job's response time is consumed by job scheduling, startup, and management overhead.

Using Nitro, a user can submit a single "Nitro job" with the 50,000 HTC jobs (now referred to as Nitro tasks) to the system's scheduler. Nitro will quickly execute this workload using its very low scheduling overhead and very quick workload management. In other words, Nitro's low scheduling overhead provides improved response time for executing many short jobs when compared to a normal scheduler.

For example, let's look at a Nitro demonstration with an investment trading enterprise. The investment trading enterprise normally submitted a workload of 10,000 of its own HTC jobs to a commercial scheduler that took 110 seconds to execute the 10,000 jobs on ten 12-core hosts. Submitted as a single Nitro job to the same commercial scheduler, a very early version of Nitro took only 9 seconds to execute the same 10,000 jobs (again now called tasks) on the same ten 12-cores hosts. Resulting in a response time speedup of 12x!

Large Queues

Let's say an HPC cluster or a datacenter has a large job queue where a significant portion of the job queue consists of HTC workloads (thousands to millions of short jobs taking seconds to minutes to complete). This causes reduced job response times from the system scheduler.

However, by using Nitro, users can consolidate these workloads into a few Nitro jobs (tens or hundreds) to improve the system scheduler's job response times. Those Nitro jobs will execute the HTC workloads on the hosts the scheduler allocates to the Nitro jobs, thereby speeding their own workload turnarounds as well as improving the other users' job turnaround times due to a large reduction in the queue size.

For example, let's say a scheduler has a job queue containing 100,000 jobs and of those 95,000 can be consolidated into 95 Nitro jobs (each executing 1,000 tasks). By consolidating, the scheduler's job queue will drop from 100,000 jobs to 5,095 jobs, which the scheduler can process in 5% of the time required for processing 100,000 jobs, for a 20x scheduling cycle speedup. This speedup also benefits all the other jobs in the queue.

Multi-threaded HTC Applications

Let's say a user has a multi-threaded application that runs on a single host and wants to run the same application many times, but each time with different parameters or datasets. The user will either submit a single job that executes the application one instance at a time with the different parameters or datasets, or submits many jobs that execute the application only once, each with different parameters or datasets. The former method does not take advantage of the many available hosts on which many application instances can execute simultaneously for a shorter overall response time but does have the advantage of not affecting the system scheduler's scheduling time. The latter method has a greater potential for a shorter overall response time but has the disadvantage of affecting the system scheduler's job throughput.

Using Nitro, the user can run multiple instances of an application on the hosts the system scheduler allocates to the Nitro job without affecting the system scheduler's scheduling time or throughput. In addition, the Nitro job will usually execute all instances in a shorter time than the system scheduler could due to much lower overhead since Nitro's orientation and optimization quickly starts the next application instance as soon as an existing application instance completes.

Regression Testing

Let's say an institution performs large quantities of regression tests each night for the applications it develops. Many regression tests are similar and can execute independently of each other. The regression test framework submits the tests as individual jobs to the system scheduler, which means many jobs for it to schedule.

Using Nitro can increase the regression test throughput due to its much lower workload management overhead; that is, it can immediately start the next regression workload as soon as an existing workload completes without the overhead a system scheduler incurs communicating job completion, scheduling the next job to start, and then starting the next job.

Theory of Operation

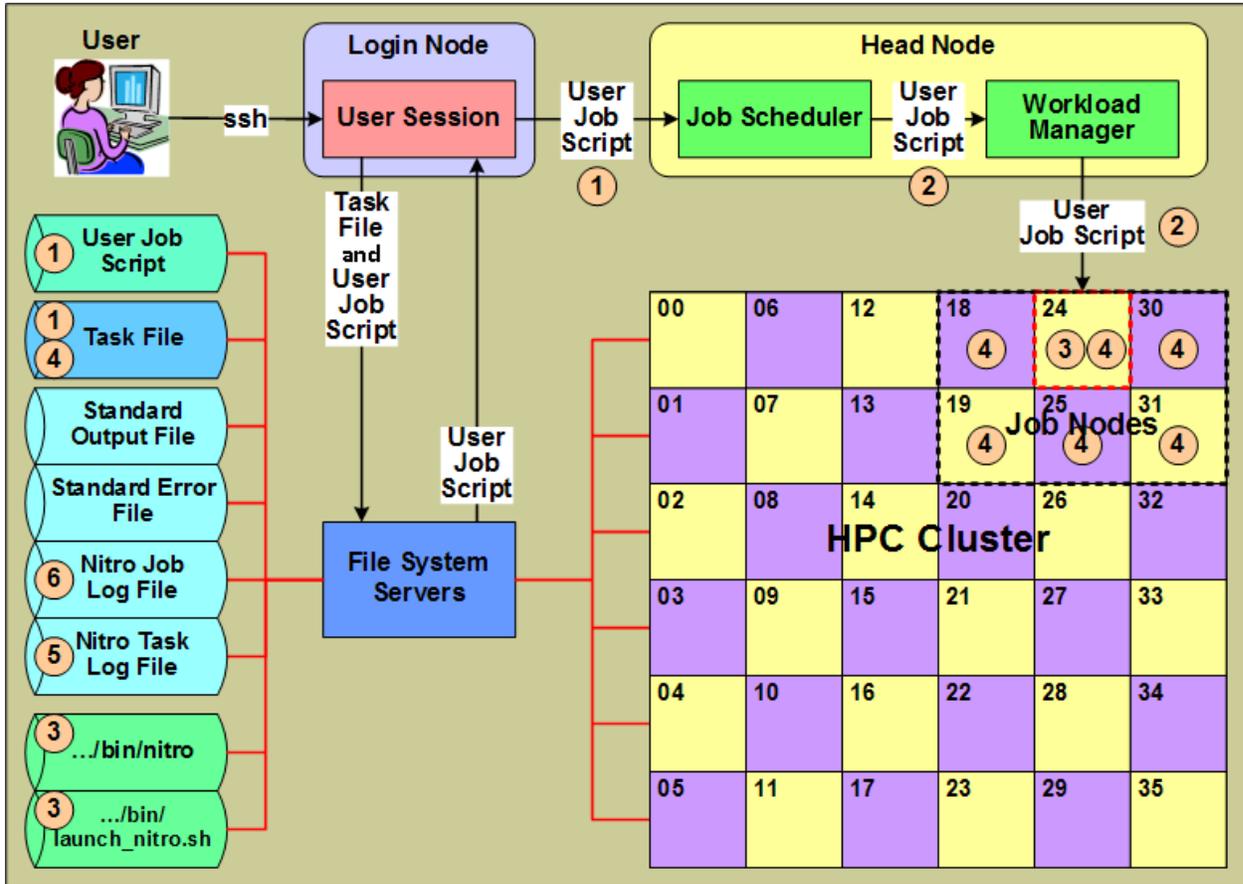
This topic identifies the various components of Nitro, describes their purpose, and illustrates how they interact with the user, the system scheduler, the system hardware, and each other.

In this topic:

- [Nitro High-Level Architecture and Flow on page 6](#)
- [Nitro Job Startup Architecture and Flow on page 8](#)
- [Nitro Job Architecture and Processing Flow on page 9](#)

Nitro High-Level Architecture and Flow

This section identifies Nitro components from a high-level Nitro product architecture perspective that most closely aligns with the perspective of a user submitting a Nitro job that uses the Nitro application to execute workloads with minimal scheduling overhead.

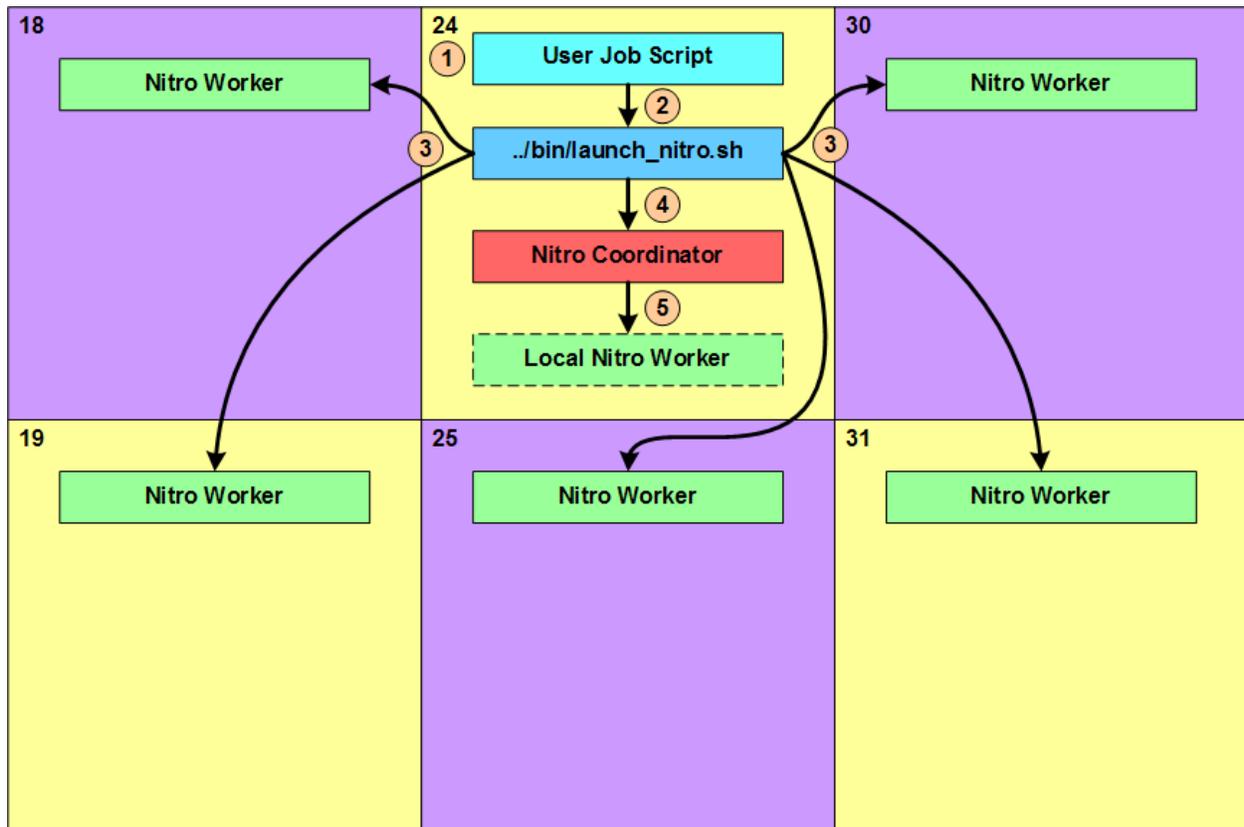


1. A user creates a "user job script" that executes Nitro with a task file (containing task definitions created by the user) that execute the user's workloads, and submits the job script to the system's job scheduler. The user job script can be scheduler-agnostic, which means the user can submit the same script to different schedulers on other systems or on the same system if its scheduler changes.
2. The scheduler allocates hosts to the Nitro job and, using a workload and/or resource manager, starts the execution of the job script on one of the job's allocated hosts.
3. The user job script executes Nitro (`../bin/nitro`) using the Nitro launch script (`../bin/launch_nitro.sh`). The Nitro launch script is scheduler-specific and allows the user job script to be scheduler-agnostic.
4. Nitro reads the task file containing the user-defined task definitions and then executes the tasks on its allocated hosts. A Nitro task definition is the equivalent of an HTC job running an application in that the user converts individual HTC jobs previously submitted to a scheduler into tasks executed by Nitro.
5. As tasks complete their execution, Nitro records information for each task in the Nitro task log file.
6. As Nitro processes task definitions from the task file and executes them, Nitro periodically updates the Nitro job log file with job progress and statistical task information to keep the user informed of its progress.

Also while Nitro executes, it records information about its environment and progress in the job's standard output file. Likewise, any unusual or unexpected errors it encounters it records in the job's standard error file.

Nitro Job Startup Architecture and Flow

This section identifies Nitro components from a job-level architecture perspective and indicates which Nitro components start and interact with other components during a Nitro job startup.

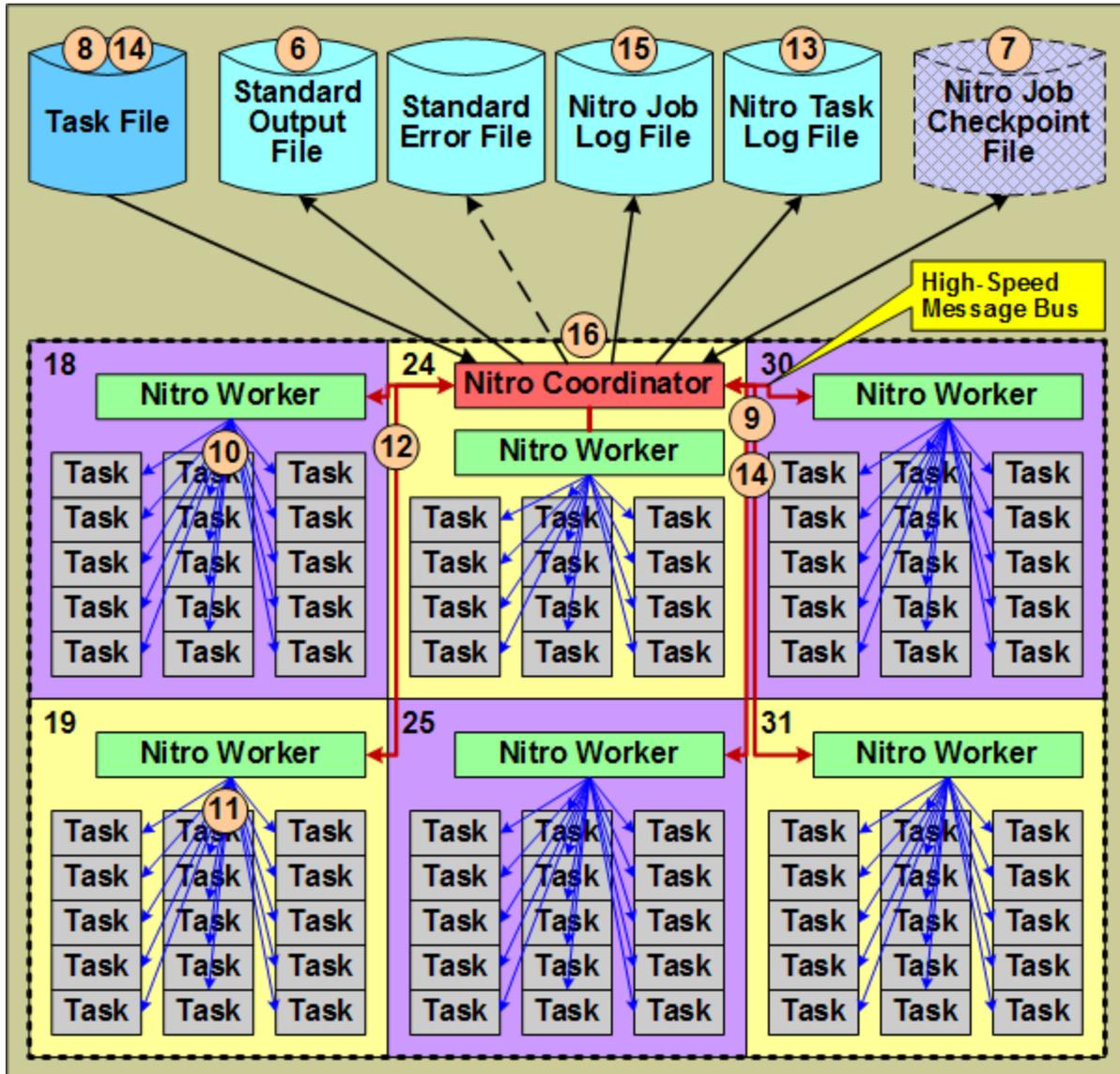


1. The user job script executes and does whatever the user specified. This can include things such as setting the Nitro environment variable for the task file path, setting other Nitro-recognized environment variables with values that affect Nitro's behavior, and performing any other preparatory work needed by the tasks Nitro will execute.
2. The last action performed by the user job script is to execute the Nitro launch script (`../bin/launch_nitro.sh`) that starts up Nitro.
3. Using scheduler- or resource manager-specific commands, the Nitro launch script starts up one Nitro worker on each of the hosts allocated to the Nitro job.
4. Lastly, the Nitro launch script starts up the Nitro coordinator on the host on which it is executing.
5. If the user job script or the Nitro launch script specified the Nitro coordinator should start up a Nitro worker (Local Nitro Worker) on its host, the coordinator does so after it starts up.

Nitro Job Architecture and Processing Flow

This section identifies the other Nitro components used and/or produced by the Nitro coordinator from a job-level architecture perspective and shows their interactions.

This section continues the job-level perspective narrative relative to Nitro's operation shown in [Nitro Job Startup Architecture and Flow on page 8](#).



6. The Nitro coordinator outputs information about its environment, including file path information, the job itself (with the job id), to the job's Standard Output File.
7. Nitro coordinator checks for the existence of a Nitro job checkpoint file. If it is present, the coordinator reads the checkpoint file and resumes the Nitro job; otherwise, the Nitro coordinator creates a checkpoint file for the Nitro job.

i A user can take advantage of restarting a Nitro job from where it left off if the user or an administrator cancelled the job or the scheduler preempted the job.

8. The Nitro coordinator opens and starts reading the task file in the user job script or via an environment variable. If the checkpoint file already exists, the coordinator resumes reading the task file from where it left off and reassigns any uncompleted tasks to workers for execution.
9. The Nitro coordinator processes the task definitions in the task file and creates task "assignments" from the task definitions in the task file that it sends to the Nitro workers via a message bus.
10. The Nitro workers each process their own task assignment and start up tasks using their "task launch" threads. Each task launch thread starts up and executes one task.
11. When a task finishes executing, the worker asynchronously uses the task launch thread to obtain statistical information about the task's execution.
12. When all tasks within a task assignment have completed, the worker returns the tasks' statistical information to the coordinator via the message bus and then starts processing its next task assignment.
13. The coordinator records the tasks' statistical information in the Nitro task Log file
14. The coordinator continues reading task definitions from the task file, creating task assignments, and sending the task assignments to the workers. To keep the workers busy (fully utilizing the hosts allocated to the Nitro job), the coordinator sends another task assignment to a worker when the worker processed a majority of the current task assignment (while worker still has tasks in its queue). This overlapping of task assignments keeps all host cores executing workload for a very high percentage of the time.
15. The coordinator periodically updates the Nitro job's statistical information in the Nitro job log file. The user can refer to the job log file to follow the Nitro job's progress.
16. When the coordinator has reached the end of the task file *and* the workers have executed all tasks, the coordinator shuts down the workers, deletes the checkpoint file, and then terminates itself. At this point the user job completes.

Nitro and System Scheduler Policies

Nitro is an HTC scheduler application executed by a job script just like any other application a user might execute. By design, Nitro does not and will not perform scheduling functions like a regular system scheduler, such as, enforcing per-user, per-group, and/or per-account resource or usage limit policies, supporting fair-share policies, maintaining separation between users in a multi-tenancy environment, etc. This means Nitro will not perform functions such as launching tasks from one Nitro job as different users, etc.

Nitro and Users

If users are subject to different constraints and policies, requiring the users to submit their own Nitro jobs and not combine the Nitro tasks allows the system scheduler to enforce policies on the

users, which is the system scheduler's responsibility. Thus with Nitro, the administrator will continue to use the system scheduler to enforce policies in the normal manner.

For example, if three users (A, B and C) each want to execute Nitro tasks that are the same application and perhaps even use the same data, they each must submit their own Nitro job with its own task file (could be the same file but the users must specify it for their own job). If they are each subject to different policies and constraints, the system scheduler enforces those constraints and, by design, Nitro is ignorant of the policies and constraints and will remain so.

Nitro and Host Resources

If a user wants to execute workloads that require certain hardware resources or constraints, the user must use the system scheduler's resource request capabilities to allocate or constrain such.

For example if a Nitro job will execute workloads that require an accelerator, such as NVIDIA/AMD GPU or Intel MIC (Xeon Phi), the user must request hosts with the required accelerator(s) for the Nitro job so the workloads have accelerator(s) available to them.

Key Terminology and Usage

This document includes a glossary of key terms used through this guide. This is to help simplify and clarify the information presented.

For example, instead of using terms specific to the traditional HPC (research, university, and government institutions) and commercial enterprise markets, such as "HPC cluster" and "datacenter" and their corresponding "compute node" and "server" terms, this guide uses the generic terms "system" and "host", respectively. Also this guide uses the term "workload" to represent an arbitrary amount of work to execute on a system while "job" refers to workload submitted by a user to a system's scheduler for eventual execution on one or more of the system's hosts.

Refer often to [Glossary on page 92](#) for a complete list of terms used in this guide.

Chapter 2 Installation and Configuration

This chapter provides Nitro installation and configuration instructions. This chapter is designed for system administrators.

In this chapter:

- [Understand and Plan Your System Environment on page 13](#)
- [System Requirements on page 14](#)
- [Manual Installation and Upgrade on page 15](#)
- [RPM Installation and Upgrade on page 36](#)

Understand and Plan Your System Environment

This topic provides information that is important to understand before you begin your Nitro installation.

In this topic:

- [Nitro Licenses and Licensing on page 13](#)
- [Nitro Product Packaging on page 13](#)
- [Default Installation Directory and Subdirectories on page 14](#)

Nitro Licenses and Licensing

Nitro is a licensed software product that requires licenses conveyed via a license file and managed by the RLM license server in order to execute. Adaptive Computing must generate a license file for your Nitro product and you must install the license file onto the RLM server where the RLM license daemon can read it. Without the proper installation of the license file, Nitro will not execute.

Adaptive Computing licenses Nitro in two mutually-exclusive ways, by node and by core.

- Sites running parallel jobs that request multiple whole host systems (HPC compute nodes or servers) will want to license Nitro by node since their schedulers often allocate job resources by whole hosts for speed.
- Sites running mainly serial (single-core) jobs that request one or a few cores will want to license Nitro by core since their schedulers tend to fragment host system resources such that it may be very difficult for a Nitro job to obtain whole host systems.

The Nitro license file indicates the license model, by-node or by-core, the Nitro product will operate under.

Nitro Product Packaging

With Nitro 2.0, you will need access to an RLM Server for licensing.

A Nitro Web Services interface is also introduced with 2.0. Moab Viewpoint requires Nitro Web Services to track Nitro job status.

Depending on your system configuration, you may need some or all of the components. The Nitro Download page includes bundles that let you download all the components.

 Nitro Web Services is currently not available for SUSE 11-based systems; it is not in the Tarball or RPM bundles for that OS.

 The RLM Server is included in the bundles as a courtesy. If your company already has access to an RLM (for example, you installed one as part of your Remote Visualization package), you will not need to install another RLM Server.

Default Installation Directory and Subdirectories

Nitro will be installed to the `/opt/nitro` directory by default. This directory includes the following subdirectories.

```
# cd /opt/nitro
# ls -l
drwxr-xr-x. 2 root root 4096 Jun  9 16:20 bin
drwxr-xr-x. 6 root root 4096 Jun  9 16:20 scripts
drwxr-xr-x. 1 root root 4096 Jun  9 16:20 etc
```

- The `bin` directory contains the binary executable program files and executable scripts that make up the Nitro product for your site.
- The `scripts` directory contains reference launch `_nitro.sh` scripts for different job schedulers and resource managers, which include Cray ALPS with Torque or Slurm, Platform LSF, Slurm, and Torque (Moab and Maui schedulers).
- The `etc` directory contains the `nitro.cfg` file. Nitro has some default settings that can be changed by setting values in the `/opt/nitro/etc/nitro.cfg` file. See [Nitro Configuration File on page 79](#) for more information.

System Requirements

This topic identifies the system requirements for your Nitro installation.

In this topic:

- [Hardware Requirements on page 14](#)
- [Supported Operating Systems on page 15](#)
- [Reprise License Manger Server Requirements on page 15](#)
- [Software Requirements on page 15](#)

Hardware Requirements

- Nitro requires one or more multi-core processors per host. Generally the more processors (sockets) and/or OS cores a host has, the more tasks Nitro can execute simultaneously on

each host; although this will be application-dependent.

- It is recommended that hosts should have sufficient memory to execute as many applications as possible so that Nitro can run them at a rate of one application instance per OS core (especially if they are not multi-threaded). This eliminates the need for users to have to request memory in their Nitro task definitions.

i See [Task File on page 86](#) for more information on specifying memory requirements.

Supported Operating Systems

Nitro supports these operating systems:

- CentOS 6.x, 7.x
- Red Hat 6.x, 7.x
- Scientific Linux 6.x, 7.x
- SUSE Linux Enterprise Server 11, 12

i SUSE Linux Enterprise Server (SLES) 11 is not available as an RPM installation method.

Reprise License Manger Server Requirements

As of version 2.0, Nitro is a licensed software product that requires the Reprise License Manager (RLM) software to check out and check in Nitro licenses. The RLM web server/license daemon software must execute on a central server accessible by all host systems on which Nitro will execute.

The RLM software itself is extremely light-weight and does not require its own server. Meaning that you can install the RLM server on the same host as your job scheduler or resource manager server.

i If your company does not already utilize an RLM, you will need to install one. See the RLM Installation documentation available from the [Nitro download site](http://www.adaptivecomputing.com/support/download-center/nitro/) (<http://www.adaptivecomputing.com/support/download-center/nitro/>).

Software Requirements

Nitro is built with all needed libraries statically linked. This provides for a quick and simple installation and helps avoid troublesome library mismatches. No additional packages need to be installed on the compute nodes.

However, users running the nitrostat utility require Python 2.6.6 or later on the system from which they are running it.

Manual Installation and Upgrade

This section provides installation, configuration, and upgrading information using the Manual Installation method.

In this section:

- [Preparing for Manual Installation or Upgrade on page 16](#)
- Install
 - [Installing RLM Server on page 17](#)
 - [Installing Nitro on page 20](#)
 - [Installing Nitro Web Services on page 24](#)
- Upgrade
 - [Upgrading Nitro on page 33](#)

Preparing for Manual Installation or Upgrade

This topic contains instructions on how to download and unpack the Nitro Tarball Bundle for all the hosts in your configuration.

 Whether you are installing tarballs on one host or on several hosts, each host (physical machine) on which a server is installed (RLM Server, Nitro, Nitro Web Services) *must* have the Nitro Tarball Bundle.

 Nitro Web Services is currently not available for SUSE 11-based systems; it is not in the Tarball bundle for that OS.

Set Up Proxies

If your site uses a proxy to connect to the Internet, do the following:

```
export http_proxy=http://<proxy_server_id>:<port>
export https_proxy=http://<proxy_server_id>:<port>
```

Download and Unpack the Nitro Tarball Bundle

The Nitro Tarball Bundle contains all the tarballs available for Nitro. However, not every tarball may be installed on the same host.

On each host (physical machine), do the following:

1. Using a web browser, navigate to the [Adaptive Computing Nitro Download](http://www.adaptivecomputing.com/support/download-center/nitro/) website (http://www.adaptivecomputing.com/support/download-center/nitro/).
2. Download the Nitro Tarball Bundle (nitro-tarball-bundle-*<version>*-*<OS>*.tar.gz).

i The variable marked <version> indicates the build's version, revision, and changeset information. The variable marked <OS> indicates the OS for which the build was designed.

3. Unpack the Nitro Tarball Bundle.

```
[root]# tar xzvf nitro-tarball-bundle-<version>-<OS>.tar.gz
```

Installing

Installing RLM Server

Access to a Reprise License Manager (RLM) server is required when using Nitro.

! The RLM Server can run multiple licenses. If your company already uses an RLM Server, you do not need to install a new one for Nitro. Skip this topic and follow the instructions in [Installing Nitro on page 20](#).

This topic contains instructions on how to install an RLM Server.

In this topic:

- [Open Necessary Ports on page 17](#)
- [Install the RLM Server on page 19](#)
- [Change the Default Passwords on page 20](#)

Open Necessary Ports

If your site is running firewall software on its hosts, you will need to configure the firewall to allow connections to the necessary ports.

i These instructions assume you are using the default ports. If your configuration will use other ports, then substitute your port numbers when opening the ports.

On the RLM Server, do the following:

1. Open the RLM Server port (5053) and the RLM Web Interface port (5054).

- Red Hat 6-based systems using iptables

```
[root]# iptables-save > /tmp/iptables.mod
[root]# vi /tmp/iptables.mod

# Add the following lines immediately *before* the line matching
# "-A INPUT -j REJECT --reject-with icmp-host-prohibited"

-A INPUT -p tcp --dport 5053:5054 -j ACCEPT

[root]# iptables-restore < /tmp/iptables.mod
[root]# service iptables save
```

- Red Hat 7-based systems using firewalld

```
[root]# firewall-cmd --add-port=5053-5054/tcp --permanent
[root]# firewall-cmd --reload
```

- SUSE 11-based systems using SuSEfirewall2

```
[root]# vi /etc/sysconfig/SuSEfirewall2

FW_SERVICES_EXT_TCP="5053 5054"

[root]# service SuSEfirewall2_setup restart
```

- SUSE 12-based systems using SuSEfirewall2

```
[root]# vi /etc/sysconfig/SuSEfirewall2

FW_SERVICES_EXT_TCP="5053 5054"

[root]# service SuSEfirewall2 restart
```

2. Open the ISV adaptativeco port for the Adaptive license-enabled products (for example: 5135).

- Red Hat 6-based systems using iptables

```
[root]# iptables-save > /tmp/iptables.mod
[root]# vi /tmp/iptables.mod

# Add the following lines immediately *before* the line matching
# "-A INPUT -j REJECT --reject-with icmp-host-prohibited"

-A INPUT -p tcp --dport 5135 -j ACCEPT

[root]# iptables-restore < /tmp/iptables.mod
[root]# service iptables save
```

- Red Hat 7-based systems using firewalld

```
[root]# firewall-cmd --add-port=5135/tcp --permanent
[root]# firewall-cmd --reload
```

- SUSE 11-based systems using SuSEfirewall2

```
[root]# vi /etc/sysconfig/SuSEfirewall2
FW_SERVICES_EXT_TCP="5135"

[root]# service SuSEfirewall2_setup restart
```

- SUSE 12-based systems using SuSEfirewall2

```
[root]# vi /etc/sysconfig/SuSEfirewall2
FW_SERVICES_EXT_TCP="5135"

[root]# service SuSEfirewall2 restart
```

Install the RLM Server

i If your configuration uses firewalls, you *must also* open the necessary ports before installing Nitro. See [Open Necessary Ports on page 17](#).

On the host where the RLM Server will reside, do the following:

1. If you have not already do so, complete the steps to prepare the host. See [Preparing for Manual Installation or Upgrade on page 16](#).
2. Identify the RLM tarball (`ac-rlm-<version>.tar.gz`).
3. Create a non-root user and group (rlm is used in the example).

```
[root]# groupadd -r rlm
[root]# useradd -r -g rlm -d /opt/rlm -c "A non-root user under which to run
Reprise License Manager" rlm
```

4. Create a directory and install the tarball files in that location (we are using `/opt/rlm` as the install location in the example).

```
[root]# mkdir -p -m 0744 /opt/rlm
[root]# cd /opt/rlm
[root]# tar -xzvf /tmp/ac-rlm-<version>.tar.gz --strip-components=1
[root]# chown -R rlm:rlm /opt/rlm
```

i The `--strip-components=1` removes the `"ac-rlm-<version>/"` from the relative path so that they are extracted into the current directory.

5. Install the startup scripts.

i If you are using a user:group other than `rlm:rlm` or a location other than `/opt/rlm`, then edit the following files to reflect those changes after copying them.

- Red Hat 6-based or SUSE 11-based systems

```
[root]# cp init.d/rlm /etc/init.d
```

- Red Hat 7-based or SUSE 12-based systems

```
[root]# cp systemd/rlm.service /etc/systemd/system
```

6. Start the services and configure the RLM Server to start automatically at system reboot.

- Red Hat 6-based or SUSE 11-based systems

```
[root]# chkconfig --add rlm
[root]# chkconfig rlm on
[root]# service rlm start
```

- Red Hat 7-based or SUSE 12-based systems

```
[root]# systemctl start rlm.service
[root]# systemctl enable rlm.service
```

Change the Default Passwords

The RLM Web interface includes two usernames (admin and user) by default. These usernames have the default password "changeme!".



If you do not change this password, RLM, and Remote Visualization, will not be secure. For tips on choosing a good password, see <https://www.us-cert.gov/ncas/tips/ST04-002>.

Do the following for *both* the user and the admin usernames:

1. Using a web browser, navigate to your RLM instance (`http://<RLM_host>:5054`; where `<RLM_host>` is the IP address or name of the RLM Server Host).
2. Log in.
3. Select **Change Password** and change the password according to your password security process.

Installing Nitro

This topic contains instructions on how to install Nitro.

Nitro

- needs to be available to all of the nodes that will be used as part of the Nitro job.
- can be installed either to each node individually *or* to a shared file system that each node can access.
- can be installed to integrate with a scheduler, such as Moab Workload Manager, or without (Nitro standalone). The instructions are the same.

In this topic:

- [Obtain a Nitro License on page 21](#)
- [Open Necessary Ports on page 22](#)
- [Install Nitro on page 23](#)
- [Verify Network Communication on page 24](#)

Obtain a Nitro License

The Nitro license file is installed on an RLM Server.



These instructions assume you already have access to an RLM Server. See [Installing RLM Server on page 17](#) for instructions on how to set up a new RLM Server.

Do the following:

1. On the RLM server, obtain the hostid and hostname.

- **hostid**

```
[root]# /opt/rlm/rlmhostid
```

You should see output similar to the following.

```
rlmhostid v12.1
Copyright (C) 2006-2016, Reprise Software, Inc. All rights reserved.

Hostid of this machine: 00259096f004
```

- **hostname**

```
[root]# /opt/rlm/rlmhostid host
```

You should see output similar to the following.

```
rlmhostid v12.1
Copyright (C) 2006-2016, Reprise Software, Inc. All rights reserved.

Hostid of this machine: host=<your-host-name>
```

2. Email licenses@adaptivecomputing.com for a license and include the hostid and hostname you just obtained.
3. Adaptive Computing will generate the license and send you the Nitro license file (.lic) file in a return email.
4. On the RLM server, do the following:

- a. Download and install the license file.

```
[root]# cd /opt/rlm
[root]# chown rlm:rlm <licenseFileName>.lic
```

- b. If the RLM Server in your configuration uses a firewall, edit the license file to reference the ISV adaptiveco port for the Adaptive license-enabled products. This is the same port

number you opened during the RLM Server installation. See the instructions to open necessary ports in the [Installing RLM Server on page 17](#) (manual installation method) or [Installing RLM Server on page 39](#) (RPM installation method) for more information.

```
[root]# vi /opt/rlm/nitro.lic

ISV adaptiveco port=5135
```

The license file already references the RLM Server port (5053 by default).

i If the RLM Server in your configuration uses different ports, you will need to modify the license file to reflect the actual ports. See the instructions to open necessary ports in the [Installing RLM Server on page 17](#) (manual installation method) or [Installing RLM Server on page 39](#) (RPM installation method) for more information.

- c. If you did *not* install an RLM Server using the file available from Adaptive Computing (for example, because your system configuration already uses one), do the following:
 - i. Download the 'adaptiveco.set' file from the [Adaptive Computing Nitro Download Center](https://www.adaptivecomputing.com/support/download-center/nitro/) (<https://www.adaptivecomputing.com/support/download-center/nitro/>).
 - ii. Copy the 'adaptiveco.set' file into the same directory where the Nitro license resides (/opt/rlm).
- d. Perform a reread to update the RLM Server with your license.

```
[root]# /opt/rlm/rlmreread
```

Open Necessary Ports

Nitro uses several ports for communication between the workers and the coordinator.

The default port is 47000, and up to four ports are used in running Nitro (ports 47000-47003).

On each compute node (coordinator), open the necessary ports.

- Red Hat 6-based systems using iptables

```
[root]# iptables-save > /tmp/iptables.mod
[root]# vi /tmp/iptables.mod

# Add the following lines immediately *before* the line matching
# "-A INPUT -j REJECT --reject-with icmp-host-prohibited"

-A INPUT -p tcp --dport 47000:47003 -j ACCEPT

[root]# iptables-restore < /tmp/iptables.mod
[root]# service iptables save
```

- Red Hat 7-based systems using firewalld

```
[root]# firewall-cmd --add-port=47000-47003/tcp --permanent
[root]# firewall-cmd --reload
```

- SUSE 11-based systems using SuSEfirewall2

```
[root]# vi /etc/sysconfig/SuSEfirewall2
FW_SERVICES_EXT_TCP="47000 47001 47002 47003"
[root]# service SuSEfirewall2_setup restart
```

- SUSE 12-based systems using SuSEfirewall2

```
[root]# vi /etc/sysconfig/SuSEfirewall2
FW_SERVICES_EXT_TCP="47000 47001 47002 47003"
[root]# service SuSEfirewall2 restart
```

Install Nitro

i You *must* complete the tasks to obtain a Nitro license before installing Nitro. See [Obtain a Nitro License on page 21](#).
If your configuration uses firewalls, you *must also* open the necessary ports before installing Nitro. See [Open Necessary Ports on page 22](#).

On the host where Nitro will reside, do the following:

1. If you have not already done so, complete the steps to prepare the host. See [Preparing for Manual Installation or Upgrade on page 16](#).
2. Change the directory to the root of the unpacked Nitro tarball bundle.

```
[root]# cd nitro-tarball-bundle-<version>-<OS>
```

3. Identify the Nitro product tarball (nitro-<version>-<OS>.tar.gz).
4. Run each of the following commands in order.

```
[root]# mkdir /opt/nitro
[root]# tar xzvpf nitro-<version>-<OS>.tar.gz -C /opt/nitro --strip-components=1
```

5. Copy the license file you generated earlier in this topic to each compute node (coordinator). On each compute node, *or* on the shared file system, do the following:

```
[root]# cp <licenseFileName>.lic /opt/nitro/bin/
```

6. Identify the `launch_nitro.sh` script version for your resource manager. This script will be copied to the bin directory from where user job scripts will execute Nitro. See [Default Installation Directory and Subdirectories on page 14](#) for more information.

Reference scripts are provided in `/opt/nitro/scripts`.

```
[root]# find /opt/nitro -name launch_nitro.sh
./scripts/lsf/launch_nitro.sh
./scripts/torque/launch_nitro.sh
./scripts/slurm/launch_nitro.sh
./scripts/alps/torque/launch_nitro.sh
./scripts/alps/slurm/launch_nitro.sh
```

- Copy the launch script to the bin directory. (This example uses the Torque-based launch script.)

```
[root]# cp /opt/nitro/scripts/torque/launch_nitro.sh /opt/nitro/bin/
```

i This is a "copy" file operation and not a "move" operation. This allows you to customize your version of the script and always have the factory version available for consultation and/or comparison.

- Customize the bin/launch_nitro.sh script as needed for your site's administrative policies. For example, to enable the Nitro coordinator's host to always execute a local Nitro worker, modify the bin/launch_nitro.sh script version to always pass the --run-local-worker command line option to the coordinator. See [Launch Scripts on page 57](#) for more information on editing the launch script.
- If you are *not* using a shared file system, copy the Nitro installation directory to *all* hosts.

```
[root]# scp -r /opt/nitro root@host002:/opt
```

Verify Network Communication

Verify that the nodes that will be running Nitro are able to communicate with the Nitro ports *and* that the nodes are able to communicate with one another.

Installing Nitro Web Services

This topic contains instructions on how to install Nitro Web Services.



Nitro Web Services is not available for SUSE 11-based systems.

Do the following in the order presented:

- [Open Necessary Ports on page 24](#)
- [Install MongoDB on page 26](#)
- [Install and Configure Nitro Web Services on page 28](#)
- [Configure Viewpoint for Nitro Web Services on page 29](#)
- [Publish Nitro Events to Nitro Web Services on page 31](#)

Open Necessary Ports

If your site is running firewall software on its hosts, you will need to configure the firewall to allow connections to the necessary ports.

In this section:

- [Open the Tornado Web Port \(9443\) and the ZMQ Port \(47100\) on page 25](#)
- [Open the MongoDB Database Port \(27017\) on page 25](#)

Open the Tornado Web Port (9443) and the ZMQ Port (47100)

On the Nitro Web Services Host, do the following:

- Red Hat 6-based systems using iptables

```
[root]# iptables-save > /tmp/iptables.mod
[root]# vi /tmp/iptables.mod

# Add the following lines immediately *before* the line matching
# "-A INPUT -j REJECT --reject-with icmp-host-prohibited"

-A INPUT -p tcp --dport 9443 -j ACCEPT
-A INPUT -p tcp --dport 47100 -j ACCEPT

[root]# iptables-restore < /tmp/iptables.mod
[root]# service iptables save
```

- Red Hat 7-based systems using firewalld

```
[root]# firewall-cmd --add-port=9443/tcp --permanent
[root]# firewall-cmd --add-port=47100/tcp --permanent
[root]# firewall-cmd --reload
```

- SUSE 12-based systems using SuSEfirewall2

```
[root]# vi /etc/sysconfig/SuSEfirewall2

FW_SERVICES_EXT_TCP="9443 47100"

[root]# service SuSEfirewall2 restart
```

Open the MongoDB Database Port (27017)

i Nitro Web Services requires access to a MongoDB database. Depending on your system configuration, your MongoDB databases may not be installed on the same host as their corresponding component servers. For example, you may choose to install the Nitro Web Services MongoDB on the same host where you have installed other MongoDB databases.

Do the following, as needed:

- If you have chosen to install the Nitro Web Services MongoDB database on the *same* host you installed other MongoDB databases, confirm the firewall port (27017) is already opened on that host.
- If you have chosen to install the Nitro Web Services MongoDB database on a *different* host from other MongoDB databases, you will need to open the Nitro Web Services MongoDB database port in firewall for that host. To open the port in the firewall, do the following:

- Red Hat 6-based systems using iptables

```
[root]# iptables-save > /tmp/iptables.mod
[root]# vi /tmp/iptables.mod

# Add the following lines immediately *before* the line matching
# "-A INPUT -j REJECT --reject-with icmp-host-prohibited"

-A INPUT -p tcp --dport 27017 -j ACCEPT

[root]# iptables-restore < /tmp/iptables.mod
[root]# service iptables save
```

- Red Hat 7-based systems using firewalld

```
[root]# firewall-cmd --add-port=27017/tcp --permanent
[root]# firewall-cmd --reload
```

- SUSE 12-based systems using SuSEfirewall2

```
[root]# vi /etc/sysconfig/SuSEfirewall2

FW_SERVICES_EXT_TCP="27017"

[root]# service SuSEfirewall2 restart
```

Install MongoDB

To install and enable MongoDB on the Nitro Web Services Host, do the following:

1. Install MongoDB.

- Red Hat 6-based or Red Hat 7-based systems

```
[root]# cat > /etc/yum.repos.d/mongodb.repo <<End-of-file
[mongodb]
name=MongoDB Repository
baseurl=http://downloads-distro.mongodb.org/repo/redhat/os/x86_64
gpgcheck=0
enabled=1
exclude=mongodb-org mongodb-org-server
End-of-file
[root]# yum install mongo-10gen-server
```

- SUSE 12-based systems

```
[root]# zypper ar --refresh -r
http://download.opensuse.org/repositories/server:/database/SLE_
12/server:database.repo
[root]# zypper install mongodb
```

2. Start MongoDB.

 There may be a short delay (approximately 3 minutes) for Mongo to start the first time.

- Red Hat 6-based systems

```
[root]# chkconfig mongod on
[root]# service mongod start
```

- Red Hat 7-based systems

```
[root]# cat > /usr/lib/systemd/system/mongodb.service <<End-of-file
[Unit]
Description=High-performance, schema-free document-oriented database
After=syslog.target network.target

[Service]
Type=forking
User=mongod
Group=mongod
Environment=CONFIG=/etc/mongod.conf
Environment=OPTIONS=
EnvironmentFile=-/etc/sysconfig/mongod
ExecStart=/usr/bin/mongod -f \${CONFIG} \${OPTIONS}
PrivateTmp=true
LimitNOFILE=65536
TimeoutStartSec=180
StandardOutput=syslog
StandardError=syslog

[Install]
WantedBy=multi-user.target
End-of-file
[root]# rm -f /etc/init.d/mongod
[root]# systemctl enable mongodb.service
[root]# systemctl start mongodb.service
[root]# systemctl daemon-reload
```

- SUSE 12-based systems

```
[root]# systemctl enable mongodb.service
[root]# systemctl start mongodb.service
[root]# systemctl daemon-reload
```

3. Prepare the MongoDB database by doing the following:

- a. Add the required MongoDB users.

i The password used below (secret1) is an example. Choose your own password for this user.

```
[root]# mongo
> use admin;
> db.addUser("admin_user", "secret1");
> db.auth ("admin_user", "secret1");
> exit
```

i Because the `admin_user` has read and write rights to the `admin` database, it also has read and write rights to all other databases. See [Control Access to MongoDB Instances with Authentication](http://docs.mongodb.org/manual/tutorial/control-access-to-mongodb-with-authentication/) (<http://docs.mongodb.org/manual/tutorial/control-access-to-mongodb-with-authentication/>) for more information.

b. Enable authentication in MongoDB.

- Red Hat 6-based systems

```
[root]# vi /etc/mongod.conf
auth = true
[root]# service mongod restart
```

- Red Hat 7-based systems

```
[root]# vi /etc/mongod.conf
auth = true
[root]# systemctl restart mongod.service
```

- SUSE 12-based systems

MongoDB authentication is already enabled. No further action is needed.

Install and Configure Nitro Web Services

i You *must* complete the tasks earlier in this topic before installing Nitro Web Services.

On the host where Nitro Web Services will reside, do the following:

1. If you have not already do so, complete the steps to prepare the host. See [Preparing for Manual Installation or Upgrade on page 16](#).
2. Identify and unpack the Nitro Web Services tarball (`nitro-web-services-<version>-<OS>.tar.gz`).

```
[root]# tar -xvzpf nitro-web-services-<version>-<OS>.tar.gz
```

3. Install Nitro Web Services.

```
[root]# cd /opt/nitro-web-services-<version>
[root]# ./install <directory>
# <directory> is optional; defaults to /opt/nitro-web-services when not given
```

4. Understand and edit the configuration files.

This includes clarifying what each configuration file is for and what to expect the first time the NWS service is started vs. each subsequent start.

MongoDB user, table, and index creation is performed at initial startup. Many of the options defined in the Nitro Web Service configuration files influence Mongo user/password and index creation.



Names and passwords are created *only* if they do not yet exist. Changing a password in the configuration file after initial startup will not update the password in Mongo.

The installation provides two configuration files.

- `/opt/nitro-web-services/etc/nitro.cfg`

This is the Nitro Web Services web application configuration file.

Before initial startup, set "admin_username" and "admin_password" to the MongoDB admin username and password you used when setting up MongoDB. It is also recommended that you change all other default passwords before starting Nitro Web Services. If you do not change the passwords at this point, it will be more difficult to change them later.

By default, NWS uses an auto-generated self-signed SSL certificate. However, you can use your own certfile, keyfile, and ca_certs files if you wish.

- `/opt/nitro-web-services/etc/zmq_job_status_adapter.cfg`

This is the Nitro ZMQ Job Status Adapter configuration file.

The Nitro ZMQ Job Status Adapter listens to job status updates on the ZMQ bus and publishes them to MongoDB using the Nitro Web Services REST API. The username and password must be set to a MongoDB user with write permissions. At minimum, set the password for nitro-writeonly-user to the password defined in `/opt/nitro-web-services/etc/nitro.cfg` and make sure the SSL options are set correctly based on SSL settings in `/opt/nitro-web-services/etc/nitro.cfg`.

5. Start the services and configure Nitro Web Services to start automatically at system boot.

- Red Hat 6-based systems

```
[root]# chkconfig --add nitro-web-services
[root]# chkconfig --add nitro-zmq-job-status-adapter
[root]# service nitro-web-services start
[root]# service nitro-zmq-job-status-adapter start
```

- Red Hat 7-based or SUSE 12-based systems

```
[root]# systemctl enable nitro-web-services.service
[root]# systemctl enable nitro-zmq-job-status-adapter.service
[root]# systemctl daemon-reload
[root]# systemctl start nitro-web-services.service
[root]# systemctl start nitro-zmq-job-status-adapter.service
```

Configure Viewpoint for Nitro Web Services

Do the following:

1. Using a web browser, navigate to your Viewpoint instance (`http://<server>:8081`) and then log in as the MWS administrative user (moab-admin, by default).
2. Click **Configuration** from the menu and then click **Nitro Services** from the left pane. The

following is an example of the Nitro Services Configuration page.

3. Enter the configuration information. The following table describes the required information.

Field	Description
Nitro WS URL	Hostname (or IP address) and port number for the host on which you installed Nitro Web Services. For example, <code>https://<hostname>:9443</code>
Username	Name of the user. This typically <code>nitro-readonly-user</code> .
Password	The user's password.
Trust Self Signed	Indicates whether Nitro Web Services was set up using self-signed certificates.

4. Click **TEST** to confirm the settings are correct. This confirms whether Nitro Web Services is up and receiving connections.
5. Click **SAVE**.
6. (Recommended) Use `curl` to test Nitro Web Services connectivity.

```
[root]# curl --insecure --data '{"username": "nitro-admin", "password": "ChangeMe2!"}' \
https://<hostname>:9443/auth
```

You should see output similar to the following.

```

{
  "status": 200,
  "data": {
    "nitro-key": "3e0fb95e9a0e44ae91daef4deb500dcc67a3714880e851d781512a49",
    "user": {
      "username": "nitro-admin",
      "last_updated": "2016-02-26 23:34:55.604000",
      "name": "Nitro Admin",
      "created": "2016-02-26 23:34:55.604000",
      "auth": {
        "job": [
          "read",
          "write",
          "delete"
        ],
        "user": [
          "read",
          "write",
          "delete"
        ]
      }
    }
  }
}

```

Publish Nitro Events to Nitro Web Services

You need to configure the Nitro coordinators to send job status updates to the Nitro Web Services's ZMQ Job Status Adapter. The ZMQ Job Status Adapter is responsible for reading job status updates off of the ZMQ bus and persisting them to Mongo. Nitro Web Services can then be used to access Nitro job status.

Each Nitro job has a Nitro Coordinator. Nitro Coordinators can be configured to publish job status updates to ZMQ by setting the "nws-connector-address" configuration option in Nitro's nitro.cfg file. Each compute node allocated/scheduled to a Nitro Job can play the role of a Nitro coordinator. Therefore, you must update the "nws-connector-address" in each compute node's nitro.cfg file.

i Configuring nws-connector-address is simplified if each node is sharing Nitro's configuration over a shared filesystem. If you are *not* using a shared filesystem, update the Nitro configuration on *each* compute node.

Do the following:

1. If you have not already done so, on the Nitro Web Services Host, locate the msg_port number in the /opt/nitro-web-services/etc/zmq_job_status_adapter.cfg file. This is the port number you need to specify for the nws-connector-address.
2. On *each* Nitro compute note (Torque MOM Host), specify the nws-connector-address in the /opt/nitro/etc/nitro.cfg file.

```

...
# Viewpoint connection allows Nitro to communicate job status information
# to viewpoint. This option indicates name and port of the remote server
# in the form: <host>:<port>
nws-connector-address <nitro-web-services-hostname>:47100
...

```

Upgrading

Upgrading RLM Server

This topic contains instructions on how to upgrade the RLM Server.

 The RLM v12.1 (build:2) release resolved memory leak and security issues. The RLM package available with Moab HPC Suite 9.0.2, contains the v12.1 (build:2) release. Adaptive Computing *strongly* recommends that your RLM Server is v12.1 (build:2).

Upgrade the RLM Server

 These instructions assume you used /opt/rlm as the install location.

On the RLM Server Host, do the following:

1. If you have not already done so, complete the steps to prepare the host. See [Preparing for Manual Installation or Upgrade on page 16](#).
2. Identify the RLM tarball (`ac-rlm-<version>.tar.gz`).
3. Stop the RLM service.

- Red Hat 6-based or SUSE 11-based systems

```
[root]# service rlm stop
```

- Red Hat 7-based or SUSE 12-based systems

```
[root]# systemctl stop rlm.service
```

4. Archive the existing RLM installation, including the license file(s).

```
[root]# mv /opt/rlm/ /opt/rlm-<archive_version>/
```

5. Install the new tarball files.

```
[root]# mkdir -p -m 0744 /opt/rlm
[root]# cd /opt/rlm
[root]# tar -xzvf /<unpack-directory>/ac-rlm-<version>.tar.gz --strip-components=1
[root]# chown -R rlm:rlm /opt/rlm
```

 The `--strip-components=1` removes the "`ac-rlm-<version>/`" from the relative path so that they are extracted into the current directory.

6. Install the startup scripts.

i If you are using a user:group other than rlm:rlm or a location other than /opt/rlm, then edit the following files to reflect those changes after copying them.

- Red Hat 6-based or SUSE 11-based systems

```
[root]# cp init.d/rlm /etc/init.d
```

- Red Hat 7-based or SUSE 12-based systems

```
[root]# cp systemd/rlm.service /etc/systemd/system
```

7. Restore the license file(s).

```
[root]# cp /opt/rlm-<archive_version>/*.lic /opt/rlm/
```

8. Restart the RLM service.

- Red Hat 6-based or SUSE 11-based systems

```
[root]# service rlm restart
```

- Red Hat 7-based or SUSE 12-based systems

```
[root]# systemctl daemon-reload
[root]# systemctl restart rlm.service
```

Upgrading Nitro

This topic contains the steps and procedures to follow to upgrade Nitro using the Manual upgrade method.

In this topic:

- [Upgrade from a Version Prior to 2.0 on page 33](#)
- [Upgrade Nitro on page 34](#)
- [Verify Network Communication on page 35](#)

Upgrade from a Version Prior to 2.0

i Beginning with Nitro 2.0, the licensing procedure changed to use an RLM server. If your company already uses an RLM Server, you can skip this procedure.

The following steps are required if you are upgrading a Nitro version prior to 2.0.

1. Install or obtain access to an RLM server. See [Installing RLM Server on page 17](#).

i Beginning with Nitro 2.0, the licensing procedure changed to use an RLM server. If your company already uses an RLM Server, you can skip this procedure.

2. Obtain and install the Nitro license. This requires access to an RLM server. See [Obtain a Nitro License on page 21](#).
3. Copy the license file to each compute node (coordinator). On each compute node, *or* on the shared file system, do the following:

```
[root]# cp <licenseFileName>.lic /opt/nitro/bin/
```

Upgrade Nitro

On the host where Nitro resides, do the following:

1. If you have not already done so, complete the steps to prepare the host. See [Preparing for Manual Installation or Upgrade on page 16](#).
2. Back up your existing launch script in `/opt/nitro/bin/`.
3. Change the directory to the root of the unpacked Nitro tarball bundle.

```
[root]# cd nitro-tarball-bundle-<version>-<OS>
```

4. Identify the Nitro product tarball (`nitro-<version>-<OS>.tar.gz`) and unpack the tarball into the same directory you created when you first installed Nitro (for example, `/opt/nitro`).

```
[root]# tar xzvpf nitro-<version>-<OS>.tar.gz -C /opt/nitro --strip-components=1
```

5. Identify the `launch_nitro.sh` script version for your resource manager.

Reference scripts are provided in `/opt/nitro/scripts`.

```
[root]# find . -name launch_nitro.sh
./scripts/lsf/launch_nitro.sh
./scripts/torque/launch_nitro.sh
./scripts/slurm/launch_nitro.sh
./scripts/alps/torque/launch_nitro.sh
./scripts/alps/slurm/launch_nitro.sh
```

6. Copy the latest launch script to the bin directory. (This example uses the Torque-based launch script.)

```
[root]# cp /opt/nitro/scripts/torque/launch_nitro.sh /opt/nitro/bin/launch_nitro.sh
```

i This is a "copy" file operation and not a "move" operation. This allows you to customize your version of the script and always have the factory version available for consultation and/or comparison.

7. Merge any customizations from your existing `launch_nitro.sh` script into the script you just copied to the bin directory.
8. If you are not using a shared file system, copy the updated Nitro installation directory to all hosts.

Only the Nitro bin directory with its proper path is required to run Nitro jobs. This means that you only need to copy the Nitro bin directory to the other hosts.

```
[root]# scp -r /opt/nitro/bin root@host002:/opt/nitro
nitrostat                100%  12KB  12.0KB/s   00:00
launch_nitro.sh         100% 6890   6.7KB/s   00:00
nitro                   100%  15MB  14.9MB/s   00:00
```

Verify Network Communication

Verify that the nodes that will be running Nitro are able to communicate with the Nitro ports *and* that the nodes are able to communicate with one another.

Upgrading Nitro Web Services

This topic contains instructions on how to upgrade Nitro Web Services.

Upgrade Nitro Web Services

On the host where Nitro Web Services resides, do the following:

1. If you have not already done so, complete the steps to prepare the host. See [Preparing for Manual Installation or Upgrade on page 16](#).
2. Stop the services.
 - Red Hat 6-based systems

```
[root]# service nitro-web-services stop
[root]# service nitro-zmq-job-status-adapter stop
```

- Red Hat 7-based or SUSE 12-based systems

```
[root]# systemctl stop nitro-web-services.service
[root]# systemctl stop nitro-zmq-job-status-adapter.service
```

3. Back up the contents of the `/opt/nitro-web-services/etc` directory (at least the `nitro.cfg` and the `zmq_job_status_adapter.cfg` files).
4. Remove the `/opt/nitro-web-services` directory.

```
[root]# rm -rf /opt/nitro-web-services
```

5. Change the directory to the root of the unpacked Nitro tarball bundle.

```
[root]# cd nitro-tarball-bundle-<version>-<OS>
```

6. Identify and unpack the Nitro Web Services tarball (nitro-web-services-*<version>*.*<OS>*.tar.gz).

```
[root]# tar xzvpf nitro-web-services-<version>.<OS>.tar.gz
```

7. Install Nitro Web Services.

```
[root]# cd ./nitro-web-services-<version>.<OS>
[root]# ./install <directory>
# <directory> is where you want to install Nitro Web Services (defaults to /opt)
```

8. Restore the nitro.cfg and the zmq_job_status_adapter.cfg files (and any other files) you backed up earlier in this procedure.

i See "[Understand and edit the configuration files. on page 28](#)" for more information on the configuration files.

9. Start the services.

- Red Hat 6-based systems

```
[root]# service nitro-web-services start
[root]# service nitro-zmq-job-status-adapter start
```

- Red Hat 7-based or SUSE 12-based systems

```
[root]# systemctl start nitro-web-services.service
[root]# systemctl start nitro-zmq-job-status-adapter.service
```

RPM Installation and Upgrade

This section provides installation, configuration, and upgrading information using the RPM Installation method.

i The RPM Installation method is not applicable for SUSE 11-based systems.

In this section:

- [Preparing for RPM Installation or Upgrade on page 37](#)
- Install
 - [Installing RLM Server on page 39](#)
 - [Installing Nitro on page 41](#)
 - [Installing Nitro Web Services on page 45](#)
- Upgrade
 - [Upgrading Nitro on page 53](#)

Preparing for RPM Installation or Upgrade

This topic contains instructions on how to download the Nitro RPM Bundle and enable the Adaptive Computing repository for all the hosts in your configuration.

i Whether you are installing or upgrading RPMs on one host or on several hosts, each host (physical machine) on which a server is installed (RLM Server, Nitro, Nitro Web Services) *must* have the Adaptive Computing Package Repository enabled.

! Nitro Web Services is currently not available for SUSE 11-based systems; it is not in the RPM bundle for that OS.

Set Up Proxies

If your site uses a proxy to connect to the Internet, do the following:

```
export http_proxy=http://<proxy_server_id>:<port>
export https_proxy=http://<proxy_server_id>:<port>
```

Enable the Adaptive Computing Package Repository

The Nitro RPM Bundle contains all the RPMs for Nitro. However, not every RPM may be installed on the same host.

On each host (physical machine), do the following:

1. Download the Nitro 2.0.1 RPM Bundle from the [Adaptive Computing](#) website.
2. Untar the Nitro RPM bundle.

```
[root]# tar xzf nitro-rpm-bundle-<version>-<OS>.tar.gz
```

3. Change directories into the untarred directory.

i Consider reviewing the README file for additional details on using the RPM distribution tarball.

4. Install the suite repositories. The `-y` option installs with the default settings for the RPM suite.

i For a description of the options of the repository installer script, run:

```
[root]# ./install-rpm-repos.sh -h
```

```
[root]# ./install-rpm-repos.sh [<repository-directory>] [-y]
```

i If the installation returns the following warning line:
 Warning: RPMDB altered outside of yum.
 This is normal and can safely be ignored.

The [*<repository-directory>*] option is the directory where you want to copy the RPMs. If no argument is given, run "install-rpm-repos.sh -h" to view usage information and identify the default directory location. If the [*<repository-directory>*] already exists, RPMs will be added to the existing directory. No files are overwritten in [*<repository-directory>*].

A repository file is also created and points to the [*<repository-directory>*] location. For Red Hat 6-based or Red Hat 7-based systems, the repository file is created in /etc/yum.repos.d/. For SUSE 12-based systems, the repository files is created in /etc/zypp/repos.d/.

For ease in repository maintenance, the install script fails if Adaptive Computing RPMs are copied to different directories. If a non-default [*<repository-directory>*] is specified, please use the same directory for future updates.

The script installs the `createrepo` package and its dependencies. You must answer "y" to all the questions in order for the RPM install of the suite to work.

Additionally, the script installs:

- The EPEL and 10gen repositories on Red Hat 6-based and Red Hat 7-based systems.
- The openSUSE Apache:Modules, devel:languages:python, devel:languages:perl, and server:database repositories on SUSE 12-based systems.

5. Test the repository.

- Red-Hat 6-based or Red Hat 7-based systems

```
[root]# yum search nitro
```

- SUSE 12-based systems

```
[root]# zypper search nitro
```

If no error is given, the repository is correctly installed. The following is an example of the output after verifying the repository.

```
Loaded plugins: fastestmirror, security
Loading mirror speeds from cached hostfile
*epel: linux.mirrors.es.net
===== N/S Matched: nitro
=====
nitro.x86_64 : Adaptive Nitro for High Throughput Computing
nitro-web-services.x86_64 : Nitro Web Services
```

Installing

Installing RLM Server

Access to a Reprise License Manager (RLM) server is required when using Nitro.



The RLM Server can run multiple licenses. If your company already uses an RLM Server, you do not need to install a new one for Nitro. Skip this topic and follow the instructions in [Installing Nitro on page 41](#).

This topic contains instructions on how to install an RLM Server.

In this topic:

- [Open Necessary Ports on page 39](#)
- [Install the RLM Server on page 40](#)
- [Change the Default Passwords on page 41](#)

Open Necessary Ports

If your site is running firewall software on its hosts, you will need to configure the firewall to allow connections to the necessary ports.



These instructions assume you are using the default ports. If your configuration will use other ports, then substitute your port numbers when opening the ports.

On the RLM Server, do the following:

1. Open the RLM Server port (5053) and the RLM Web Interface port (5054).

- Red Hat 6-based systems using iptables

```
[root]# iptables-save > /tmp/iptables.mod
[root]# vi /tmp/iptables.mod

# Add the following lines immediately *before* the line matching
# "-A INPUT -j REJECT --reject-with icmp-host-prohibited"

-A INPUT -p tcp --dport 5053:5054 -j ACCEPT

[root]# iptables-restore < /tmp/iptables.mod
[root]# service iptables save
```

- Red Hat 7-based systems using firewalld

```
[root]# firewall-cmd --add-port=5053-5054/tcp --permanent
[root]# firewall-cmd --reload
```

- SUSE 12-based systems using SuSEfirewall2

```
[root]# vi /etc/sysconfig/SuSEfirewall2
FW_SERVICES_EXT_TCP="5053 5054"

[root]# service SuSEfirewall2 restart
```

2. Open the ISV adaptiveco port for the Adaptive license-enabled products (for example: 5135).

- Red Hat 6-based systems using iptables

```
[root]# iptables-save > /tmp/iptables.mod
[root]# vi /tmp/iptables.mod

# Add the following lines immediately *before* the line matching
# "-A INPUT -j REJECT --reject-with icmp-host-prohibited"

-A INPUT -p tcp --dport 5135 -j ACCEPT

[root]# iptables-restore < /tmp/iptables.mod
[root]# service iptables save
```

- Red Hat 7-based systems using firewalld

```
[root]# firewall-cmd --add-port=5135/tcp --permanent
[root]# firewall-cmd --reload
```

- SUSE 12-based systems using SuSEfirewall2

```
[root]# vi /etc/sysconfig/SuSEfirewall2
FW_SERVICES_EXT_TCP="5135"

[root]# service SuSEfirewall2 restart
```

Install the RLM Server

i If your configuration uses firewalls, you *must also* open the necessary ports before installing the RLM Server . See [Open Necessary Ports on page 39](#).

On the host where the RLM Server will reside, do the following:

1. If you are installing RLM Server on its own host *or* on a host that does not have another RPM installation, complete the steps to prepare the host. See [Preparing for RPM Installation or Upgrade on page 37](#).
2. Install the RPM.

- Red Hat 6-based or Red Hat 7-based systems

```
[root]# yum install ac-rlm
```

- SUSE 12-based systems

```
[root]# zypper install ac-rlm
```

Change the Default Passwords

The RLM Web interface includes two usernames (admin and user) by default. These usernames have the default password "changeme!".

 If you do not change this password, RLM will not be secure. For tips on choosing a good password, see <https://www.us-cert.gov/ncas/tips/ST04-002>.

Do the following for *both* the user and the admin usernames:

1. Using a web browser, navigate to your RLM instance (http://<RLM_host>:5054; where <RLM_host> is the IP address or name of the RLM Server Host).
2. Log in.
3. Select **Change Password** and change the password according to your password security process.

Installing Nitro

This topic contains instructions on how to install Nitro, including licensing and configuration information.

Nitro

- needs to be available to all of the nodes that will be used as part of the Nitro job.
- can be installed either to each node individually *or* to a shared file system that each node can access.
- can be installed to integrate with a scheduler, such as Moab Workload Manager, or without (Nitro standalone). The instructions are the same.

In this topic:

- [Obtain a Nitro License on page 41](#)
- [Open Necessary Ports on page 43](#)
- [Install Nitro on page 43](#)
- [Verify Network Communication on page 44](#)

Obtain a Nitro License

The Nitro license file is installed on an RLM Server.

 These instructions assume you already have access to an RLM Server. See [Installing RLM Server on page 39](#) for instructions on how to set up a new RLM Server.

Do the following:

1. On the RLM server, obtain the hostid and hostname.

- hostid

```
[root]# /opt/rlm/rlmhostid
```

You should see output similar to the following.

```
rlmhostid v12.1
Copyright (C) 2006-2016, Reprise Software, Inc. All rights reserved.

Hostid of this machine: 00259096f004
```

- hostname

```
[root]# /opt/rlm/rlmhostid host
```

You should see output similar to the following.

```
rlmhostid v12.1
Copyright (C) 2006-2016, Reprise Software, Inc. All rights reserved.

Hostid of this machine: host=<your-host-name>
```

2. Email licenses@adaptivecomputing.com for a license and include the hostid and hostname you just obtained.
3. Adaptive Computing will generate the license and send you the Nitro license file (.lic) file in a return email.
4. On the RLM server, do the following:
 - a. Download and install the license file.

```
[root]# cd /opt/rlm
[root]# chown rlm:rlm <licenseFileName>.lic
```

- b. If the RLM Server in your configuration uses a firewall, edit the license file to reference the ISV adaptiveco port for the Adaptive license-enabled products. This is the same port number you opened during the RLM Server installation. See the instructions to open necessary ports in the [Installing RLM Server on page 17](#) (manual installation method) or [Installing RLM Server on page 39](#) (RPM installation method) for more information.

```
[root]# vi /opt/rlm/nitro.lic
```

```
ISV adaptiveco port=5135
```

The license file already references the RLM Server port (5053 by default).

i If the RLM Server in your configuration uses different ports, you will need to modify the license file to reflect the actual ports. See the instructions to open necessary ports in the [Installing RLM Server on page 17](#) (manual installation method) or [Installing RLM Server on page 39](#) (RPM installation method) for more information.

- c. If you did *not* install an RLM Server using the file available from Adaptive Computing (for example, because your system configuration already uses one), do the following:
 - i. Download the 'adaptiveco.set' file from the [Adaptive Computing Nitro Download Center](https://www.adaptivecomputing.com/support/download-center/nitro/) (<https://www.adaptivecomputing.com/support/download-center/nitro/>).
 - ii. Copy the 'adaptiveco.set' file into the same directory where the Nitro license resides (/opt/rlm).
- d. Perform a reread to update the RLM Server with your license.

```
[root]# /opt/rlm/rlmreread
```

Open Necessary Ports

Nitro uses several ports for communication between the workers and the coordinator.

The default port is 47000, and up to four ports are used in running Nitro (ports 47000-47003).

On each compute node (coordinator), open the necessary ports.

- Red Hat 6-based systems using iptables

```
[root]# iptables-save > /tmp/iptables.mod
[root]# vi /tmp/iptables.mod

# Add the following lines immediately *before* the line matching
# "-A INPUT -j REJECT --reject-with icmp-host-prohibited"

-A INPUT -p tcp --dport 47000:47003 -j ACCEPT

[root]# iptables-restore < /tmp/iptables.mod
[root]# service iptables save
```

- Red Hat 7-based systems using firewalld

```
[root]# firewall-cmd --add-port=47000-47003/tcp --permanent
[root]# firewall-cmd --reload
```

- SUSE 12-based systems using SuSEfirewall2

```
[root]# vi /etc/sysconfig/SuSEfirewall2

FW_SERVICES_EXT_TCP="47000 47001 47002 47003"

[root]# service SuSEfirewall2 restart
```

Install Nitro

i You *must* complete the tasks to obtain a Nitro license before installing Nitro. See [Obtain a Nitro License on page 41](#).

If your configuration uses firewalls, you *must also* open the necessary ports before installing Nitro. See [Open Necessary Ports on page 43](#).

On the host where Nitro will reside, do the following:

1. If you are installing Nitro on its own host *or* on a host that does not have another RPM installation, complete the steps to prepare the host. See [Preparing for RPM Installation or Upgrade on page 37](#).

2. Install the RPM.

- Red Hat 6-based or Red Hat 7-based systems

```
[root]# yum install nitro
```

- SUSE 12-based systems

```
[root]# zypper install nitro
```

3. Copy the license file you generated earlier in this topic to each compute node (coordinator). On each compute node, *or* on the shared file system, do the following:

```
[root]# cp <licenseFileName>.lic /opt/nitro/bin/
```

4. Identify the `launch_nitro.sh` script version for your resource manager. This script will be copied to the bin directory from where user job scripts will execute Nitro. See [Default Installation Directory and Subdirectories on page 14](#) for more information.

Reference scripts are provided in `/opt/nitro/scripts`.

```
[root]# find /opt/nitro -name launch_nitro.sh
./scripts/lsf/launch_nitro.sh
./scripts/torque/launch_nitro.sh
./scripts/slurm/launch_nitro.sh
./scripts/alps/torque/launch_nitro.sh
./scripts/alps/slurm/launch_nitro.sh
```

5. Copy the launch script to the bin directory. (This example uses the Torque-based launch script.)

```
[root]# cp /opt/nitro/scripts/torque/launch_nitro.sh /opt/nitro/bin/
```

i This is a "copy" file operation and not a "move" operation. This allows you to customize your version of the script and always have the factory version available for consultation and/or comparison.

6. Customize the `bin/launch_nitro.sh` script as needed for your site's administrative policies. For example, to enable the Nitro coordinator's host to always execute a local Nitro worker, modify the `bin/launch_nitro.sh` script version to always pass the `--run-local-worker` command line option to the coordinator. See [Launch Scripts on page 57](#) for more information on editing the launch script.

7. If you are *not* using a shared file system, copy the Nitro installation directory to *all* hosts.

```
[root]# scp -r /opt/nitro root@host002:/opt
```

Verify Network Communication

Verify that the nodes that will be running Nitro are able to communicate with the Nitro ports *and* that the nodes are able to communicate with one another.

Installing Nitro Web Services

This topic contains instructions on how to install Nitro Web Services.



Nitro Web Services is not available for SUSE 11-based systems.

Do the following in the order presented:

1. [Open Necessary Ports on page 45](#)
2. [Install MongoDB on page 46](#)
3. [Install and Configure Nitro Web Services on page 49](#)
4. [Configure Viewpoint for Nitro Web Services on page 50](#)
5. [Publish Nitro Events to Nitro Web Services on page 52](#)

Open Necessary Ports

If your site is running firewall software on its hosts, you will need to configure the firewall to allow connections to the necessary ports.

In this section:

- [Open the Tornado Web Port \(9443\) and the ZMQ Port \(47100\) on page 45](#)
- [Open the MongoDB Database Port \(27017\) on page 46](#)

Open the Tornado Web Port (9443) and the ZMQ Port (47100)

On the Nitro Web Services Host, do the following:

- Red Hat 6-based systems using iptables

```
[root]# iptables-save > /tmp/iptables.mod
[root]# vi /tmp/iptables.mod

# Add the following lines immediately *before* the line matching
# "-A INPUT -j REJECT --reject-with icmp-host-prohibited"

-A INPUT -p tcp --dport 9443 -j ACCEPT
-A INPUT -p tcp --dport 47100 -j ACCEPT

[root]# iptables-restore < /tmp/iptables.mod
[root]# service iptables save
```

- Red Hat 7-based systems using firewalld

```
[root]# firewall-cmd --add-port=9443/tcp --permanent
[root]# firewall-cmd --add-port=47100/tcp --permanent
[root]# firewall-cmd --reload
```

- SUSE 12-based systems using SuSEfirewall2

```
[root]# vi /etc/sysconfig/SuSEfirewall2
FW_SERVICES_EXT_TCP="9443 47100"
[root]# service SuSEfirewall2 restart
```

Open the MongoDB Database Port (27017)

i Nitro Web Services requires access to a MongoDB database. Depending on your system configuration, your MongoDB databases may not be installed on the same host as their corresponding component servers. For example, you may choose to install the Nitro Web Services MongoDB on the same host where you have installed other MongoDB databases.

Do the following, as needed:

- If you have chosen to install the Nitro Web Services MongoDB database on the *same* host you installed other MongoDB databases, confirm the firewall port (27017) is already opened on that host.
- If you have chosen to install the Nitro Web Services MongoDB database on a *different* host from other MongoDB databases, you will need to open the Nitro Web Services MongoDB database port in the firewall for that host. To open the port in the firewall, do the following:
 - Red Hat 6-based systems using iptables

```
[root]# iptables-save > /tmp/iptables.mod
[root]# vi /tmp/iptables.mod

# Add the following lines immediately *before* the line matching
# "-A INPUT -j REJECT --reject-with icmp-host-prohibited"

-A INPUT -p tcp --dport 27017 -j ACCEPT

[root]# iptables-restore < /tmp/iptables.mod
[root]# service iptables save
```

- Red Hat 7-based systems using firewalld

```
[root]# firewall-cmd --add-port=27017/tcp --permanent
[root]# firewall-cmd --reload
```

- SUSE 12-based systems using SuSEfirewall2

```
[root]# vi /etc/sysconfig/SuSEfirewall2
FW_SERVICES_EXT_TCP="27017"
[root]# service SuSEfirewall2 restart
```

Install MongoDB

To install and enable MongoDB on the Nitro Web Services Host, do the following:

1. Install MongoDB.

- Red Hat 6-based or Red Hat 7-based systems

```
[root]# cat > /etc/yum.repos.d/mongodb.repo <<End-of-file
[mongodb]
name=MongoDB Repository
baseurl=http://downloads-distro.mongodb.org/repo/redhat/os/x86_64
gpgcheck=0
enabled=1
exclude=mongodb-org mongodb-org-server
End-of-file
[root]# yum install mongo-10gen-server
```

- SUSE 12-based systems

```
[root]# zypper ar --refresh -r
http://download.opensuse.org/repositories/server:/database/SLE_
12/server:database.repo
[root]# zypper install mongodb
```

2. Start MongoDB.

i There may be a short delay (approximately 3 minutes) for Mongo to start the first time.

- Red Hat 6-based systems

```
[root]# chkconfig mongod on
[root]# service mongod start
```

- Red Hat 7-based systems

```
[root]# cat > /usr/lib/systemd/system/mongodb.service <<End-of-file
[Unit]
Description=High-performance, schema-free document-oriented database
After=syslog.target network.target

[Service]
Type=forking
User=mongod
Group=mongod
Environment=CONFIG=/etc/mongod.conf
Environment=OPTIONS=
EnvironmentFile=-/etc/sysconfig/mongod
ExecStart=/usr/bin/mongod -f \${CONFIG} \${OPTIONS}
PrivateTmp=true
LimitNOFILE=65536
TimeoutStartSec=180
StandardOutput=syslog
StandardError=syslog

[Install]
WantedBy=multi-user.target
End-of-file
[root]# rm -f /etc/init.d/mongod
[root]# systemctl enable mongodb.service
[root]# systemctl start mongodb.service
[root]# systemctl daemon-reload
```

- SUSE 12-based systems

```
[root]# systemctl enable mongodb.service
[root]# systemctl start mongodb.service
[root]# systemctl daemon-reload
```

3. Prepare the MongoDB database by doing the following:

- a. Add the required MongoDB users.

i The password used below (secret1) is an example. Choose your own password for this user.

```
[root]# mongo
> use admin;
> db.addUser("admin_user", "secret1");
> db.auth("admin_user", "secret1");
> exit
```

i Because the `admin_user` has read and write rights to the `admin` database, it also has read and write rights to all other databases. See [Control Access to MongoDB Instances with Authentication](http://docs.mongodb.org/manual/tutorial/control-access-to-mongodb-with-authentication/) (<http://docs.mongodb.org/manual/tutorial/control-access-to-mongodb-with-authentication/>) for more information.

b. Enable authentication in MongoDB.

- Red Hat 6-based systems

```
[root]# vi /etc/mongod.conf
auth = true
[root]# service mongod restart
```

- Red Hat 7-based systems

```
[root]# vi /etc/mongod.conf
auth = true
[root]# systemctl restart mongod.service
```

- SUSE 12-based systems

MongoDB authentication is already enabled. No further action is needed.

Install and Configure Nitro Web Services

i You *must* complete the tasks earlier in this topic before installing Nitro Web Services.

On the host where Nitro Web Services will reside, do the following:

1. If you have not already do so, complete the steps to prepare the host. See [Preparing for RPM Installation or Upgrade on page 37](#).
2. Install the Nitro Web Services RPM.
 - Red Hat 6-based or Red Hat 7-based systems

```
[root]# yum install -y nitro-web-services
```

- SUSE 12-based systems

```
[root]# zypper --non-interactive install nitro-web-services
```

3. Understand and edit the configuration files.

This includes clarifying what each configuration file is for and what to expect the first time the NWS service is started vs. each subsequent start.

MongoDB user, table, and index creation is performed at initial startup. Many of the options defined in the Nitro Web Service configuration files influence Mongo user/password and index creation.

⚠ Usernames and passwords are created *only* if they do not yet exist. Changing a password in the configuration file after initial startup will not update the password in Mongo.

The installation provides two configuration files.

- /opt/nitro-web-services/etc/nitro.cfg

This is the Nitro Web Services web application configuration file.

Before initial startup, set "admin_username" and "admin_password" to the MongoDB admin username and password you used when setting up MongoDB. It is also recommended that you change all other default passwords before starting Nitro Web Services. If you do not change the passwords at this point, it will be more difficult to change them later.

By default, NWS uses an auto-generated self-signed SSL certificate. However, you can use your own certfile, keyfile, and ca_certs files if you wish.

- /opt/nitro-web-services/etc/zmq_job_status_adapter.cfg

This is the Nitro ZMQ Job Status Adapter configuration file.

The Nitro ZMQ Job Status Adapter listens to job status updates on the ZMQ bus and publishes them to MongoDB using the Nitro Web Services REST API. The username and password must be set to a MongoDB user with write permissions. At minimum, set the password for nitro-writeonly-user to the password defined in /opt/nitro-web-services/etc/nitro.cfg and make sure the SSL options are set correctly based on SSL settings in /opt/nitro-web-services/etc/nitro.cfg.

4. If you did not need to install the Nitro Web Services MongoDB database earlier in this topic, verify that the 'mongodb_host' and 'mongodb_port' in /opt/nitro-web-services/etc/nitro.cfg are set correctly ('localhost' on port '27017' are the defaults).
5. Start the services and configure Nitro Web Services to start automatically at system boot.
 - Red Hat 6-based systems

```
[root]# chkconfig --add nitro-web-services
[root]# chkconfig --add nitro-zmq-job-status-adapter
[root]# service nitro-web-services start
[root]# service nitro-zmq-job-status-adapter start
```

- Red Hat 7-based or SUSE 12-based systems

```
[root]# systemctl enable nitro-web-services.service
[root]# systemctl enable nitro-zmq-job-status-adapter.service
[root]# systemctl daemon-reload
[root]# systemctl start nitro-web-services.service
[root]# systemctl start nitro-zmq-job-status-adapter.service
```

Configure Viewpoint for Nitro Web Services

Do the following:

1. Using a web browser, navigate to your Viewpoint instance (`http://<server>:8081`) and then log in as the MWS administrative user (moab-admin, by default).
2. Click **Configuration** from the menu and then click **Nitro Services** from the left pane. The following is an example of the Nitro Services Configuration page.

- Enter the configuration information. The following table describes the required information.

Field	Description
Nitro WS URL	Hostname (or IP address) and port number for the host on which you installed Nitro Web Services. For example, <code>https://<hostname>:9443</code>
Username	Name of the user. This typically <code>nitro-readonly-user</code> .
Password	The user's password.
Trust Self Signed	Indicates whether Nitro Web Services was set up using self-signed certificates.

- Click **TEST** to confirm the settings are correct. This confirms whether Nitro Web Services is up and receiving connections.
- Click **SAVE**.
- (Recommended) Use `curl` to test Nitro Web Services connectivity.

```
[root]# curl --insecure --data '{"username": "nitro-admin", "password":
"ChangeMe2!"}' \
https://<hostname>:9443/auth
```

You should see output similar to the following.

```

{
  "status": 200,
  "data": {
    "nitro-key": "3e0fb95e9a0e44ae91daef4deb500dcc67a3714880e851d781512a49",
    "user": {
      "username": "nitro-admin",
      "last_updated": "2016-02-26 23:34:55.604000",
      "name": "Nitro Admin",
      "created": "2016-02-26 23:34:55.604000",
      "auth": {
        "job": [
          "read",
          "write",
          "delete"
        ],
        "user": [
          "read",
          "write",
          "delete"
        ]
      }
    }
  }
}

```

Publish Nitro Events to Nitro Web Services

You need to configure the Nitro coordinators to send job status updates to the Nitro Web Services's ZMQ Job Status Adapter. The ZMQ Job Status Adapter is responsible for reading job status updates off of the ZMQ bus and persisting them to Mongo. Nitro Web Services can then be used to access Nitro job status.

Each Nitro job has a Nitro Coordinator. Nitro Coordinators can be configured to publish job status updates to ZMQ by setting the "nws-connector-address" configuration option in Nitro's `nitro.cfg` file. Each compute node allocated/scheduled to a Nitro Job can play the role of a Nitro coordinator. Therefore, you must update the "nws-connector-address" in each compute node's `nitro.cfg` file.

i Configuring `nws-connector-address` is simplified if each node is sharing Nitro's configuration over a shared filesystem. If you are *not* using a shared filesystem, update the Nitro configuration on *each* compute node.

Do the following:

1. If you have not already done so, on the Nitro Web Services Host, locate the `msg_port` number in the `/opt/nitro-web-services/etc/zmq_job_status_adapter.cfg` file. This is the port number you need to specify for the `nws-connector-address`.
2. On *each* Nitro compute note (Torque MOM Host), specify the `nws-connector-address` in the `/opt/nitro/etc/nitro.cfg` file.

```

...
# Viewpoint connection allows Nitro to communicate job status information
# to viewpoint. This option indicates name and port of the remote server
# in the form: <host>:<port>
nws-connector-address <nitro-web-services-hostname>:47100
...

```

Upgrading

Upgrading RLM Server (RPM)

This topic contains instructions on how to upgrade the RLM Server using the RPM upgrade method.



The RLM v12.1 (build:2) release resolved memory leak and security issues. The RLM package available with Moab HPC Suite 9.0.2, contains the v12.1 (build:2) release. Adaptive Computing *strongly* recommends that your RLM Server is v12.1 (build:2).

Install the RLM Server

On the RLM Server Host, do the following:

1. If you installed the RLM Server on its own host *or* if the RLM Server is the first component being upgraded on a host with other RPM installations,, complete the steps to prepare the host. See [1.1 Preparing the Host – Typical Method](#) or [1.1 Preparing the Host – Offline Method](#).
2. Stop the RLM service.

```
[root]# service rlm stop
```

```
[root]# systemctl stop rlm.service
```

```
[root]# systemctl stop rlm.service
```

3. Install the upgrade.

```
[root]# yum update ac-rlm*
```

```
[root]# zypper update ac-rlm*
```

4. Restart the RLM service.

```
[root]# service rlm restart
```

```
[root]# systemctl daemon-reload
[root]# systemctl restart rlm.service
```

```
[root]# systemctl daemon-reload
[root]# systemctl restart rlm.service
```

Upgrading Nitro



You must be a root user when installing or upgrading Nitro.

This topic contains the steps and procedures to follow to upgrade Nitro using the RPM upgrade method.

In this topic:

- [Upgrade from a Version Prior to 2.0 on page 54](#)
- [Upgrade Nitro on page 54](#)

Upgrade from a Version Prior to 2.0

The following steps are required if you are upgrading a Nitro version prior to 2.0.

1. Install or obtain access to an RLM server. See [Installing RLM Server on page 39](#).

i Beginning with Nitro 2.0, the licensing procedure changed to use an RLM server. If your company already uses an RLM Server, you can skip this procedure.

2. Obtain and install the Nitro license. This requires access to an RLM server. See [Obtain a Nitro License on page 41](#).
3. Copy the license file to each compute node (coordinator). On each compute node, *or* on the shared file system, do the following:

```
[root]# cp <licenseFileName>.lic /opt/nitro/bin/
```

Upgrade Nitro

On the host where Nitro resides, do the following:

1. If you installed Nitro on its own host *or* if Nitro is the first component being upgraded on a host with other RPM installations, complete the steps to prepare the host. See [Preparing for RPM Installation or Upgrade on page 37](#).
2. Back up your existing launch script in `/opt/nitro/bin/`.
3. Install the upgrade.

- Red Hat 6-based or Red Hat 7-based systems

```
[root]# yum update nitro
```

- SUSE 12-based systems

```
[root]# zypper update nitro
```

4. Identify the `launch_nitro.sh` script version for your resource manager.

Reference scripts are provided in `/opt/nitro/scripts`.

```
[root]# find . -name launch_nitro.sh
./scripts/lsf/launch_nitro.sh
./scripts/torque/launch_nitro.sh
./scripts/slurm/launch_nitro.sh
./scripts/alps/torque/launch_nitro.sh
./scripts/alps/slurm/launch_nitro.sh
```

- Copy the latest launch script to the bin directory. (This example uses the Torque-based launch script.)

```
[root]# cp /opt/nitro/scripts/torque/launch_nitro.sh /opt/nitro/bin/launch_nitro.sh
```

i This is a "copy" file operation and not a "move" operation. This allows you to customize your version of the script and always have the factory version available for consultation and/or comparison.

- Merge any customizations from your existing `launch_nitro.sh` script into the script you just copied to the bin directory.
- If you are not using a shared file system, copy the updated Nitro installation directory to all hosts.

Only the Nitro bin directory with its proper path is required to run Nitro jobs. This means that you only need to copy the Nitro bin directory to the other hosts.

```
[root]# scp -r /opt/nitro/bin root@host002:/opt/nitro
nitrostat                100%  12KB  12.0KB/s   00:00
launch_nitro.sh         100% 6890    6.7KB/s   00:00
nitro                    100%  15MB  14.9MB/s   00:00
```

Upgrading Nitro Web Services (RPM)

This topic contains instructions on how to upgrade Nitro Web Services using the RPM upgrade method.

Upgrade Nitro Web Services

On the Nitro Web Services Host, do the following:

- If you installed Nitro Web Services on its own host *or* if Nitro Web Services is the first component being upgraded on a host with other RPM installations, complete the steps to prepare the host. See [1.1 Preparing the Host – Typical Method](#) or [1.1 Preparing the Host – Offline Method](#).
- Stop the services.
 - Red Hat 6-based systems

```
[root]# service nitro-web-services stop
[root]# service nitro-zmq-job-status-adapter stop
```

- Red Hat 7-based or SUSE 12-based systems

```
[root]# systemctl stop nitro-web-services.service
[root]# systemctl stop nitro-zmq-job-status-adapter.service
```

3. Install the upgrade.

- Red Hat 6-based or Red Hat 7-based systems

```
[root]# yum update nitro-web-services
```

- SUSE 12-based systems

```
[root]# zypper update nitro-web-services
```

4. If you are upgrading Nitro from 2.0.0, re-enable the services.

- Red Hat 6-based systems

```
[root]# chkconfig nitro-web-services on  
[root]# chkconfig nitro-zmq-job-status-adapter restart
```

- Red Hat 7-based or SUSE 12-based systems

```
[root]# systemctl enable nitro-web-services.service  
[root]# systemctl enable nitro-zmq-job-status-adapter.service
```

5. Start the services.

- Red Hat 6-based systems

```
[root]# service nitro-web-services start  
[root]# service nitro-zmq-job-status-adapter start
```

- Red Hat 7-based or SUSE 12-based systems

```
[root]# systemctl start nitro-web-services.service  
[root]# systemctl start nitro-zmq-job-status-adapter.service
```

Chapter 3 System Administration

This chapter provides additional configuration and resources, including troubleshooting, for system administrators.

In this chapter:

- [Scheduler and Resource Manager Integration on page 57](#)
- [File System Configuration on page 58](#)
- [Run Nitro Without a Scheduler on page 59](#)

Scheduler and Resource Manager Integration

This topic provides information on the scripts used to integrate with your resource manager. It also provides additional configuration instructions if your installation uses a directory other than the default.

In this topic:

- [Launch Scripts on page 57](#)
- [Additional Configuration on page 58](#)

Launch Scripts

The `/opt/nitro/scripts` directory contains a set of launch scripts for different job schedulers and resource managers, which include Cray ALPS with Torque or Slurm, Platform LSF, Slurm, and Torque (Moab and Maui schedulers). See [Resource Manager Launch Scripts on page 80](#).

The purpose of these scripts is to provide an interface between the job scheduler or resource manager and Nitro that supplies Nitro with information about the job and resources.

Customizing the Launch Scripts

The launch scripts may be customized for your installation. The launch script needs to accomplish the following:

1. Get a list of nodes allocated to the job.
2. Get the job ID.
3. Translate environment variables to command line options.

Users can set the environment variables `NITRO_OPTIONS` (applied to both the worker and coordinator command lines), `NITRO_WORKER_OPTIONS` (applied only to the worker command line), and `NITRO_COORD_OPTIONS` (applied only to the coordinator command line). See [Submit a Nitro Job to a Scheduler on page 67](#).

4. Launch the workers on all but the first node allocated to the job.
5. Launch the coordinator on the first node allocated to the job.

Additional Configuration

All of the scripts have a line setting the "NITRO" script variable to the Nitro default installation location.

If you installed Nitro to a directory other than the default `/opt/nitro/bin` installation directory, you will need to change the launch scripts to run Nitro from your installation location.

For example, if you have installed Nitro on a shared file system called `"/sharedfs/opt/nitro"`, then make the following change to your copy of the launch script.

change:

```
NITRO=/opt/nitro/bin/nitro
```

to:

```
NITRO=/sharedfs/opt/nitro/bin/nitro
```

File System Configuration

This topic provides details regarding the file system configuration.

Users will normally submit Nitro jobs to a scheduler. A Nitro job runs in the user's workspace on the nodes allocated to the job by the scheduler. There are two Nitro output file locations (task log and job log) that *must* be available to the hosts running the workers and coordinator.

In this topic:

- [Nitro Launch Script Location on page 58](#)
- [Task File Location on page 58](#)
- [Job Output Directory on page 58](#)
- [Recommended NFS Settings on page 59](#)

Nitro Launch Script Location

The Nitro launch script must be installed in a shared file location or a location that is accessible to the users that will be running Nitro jobs. While users won't need to run the Nitro application itself, they should have read access to the Nitro launch script.

Task File Location

The task file a Nitro job will use must be accessible to the job. Most schedulers will copy the user job script, but will not copy the task file the user must provide. Users may need to stage this task file into a location accessible to Nitro job hosts. Nitro only needs read access to this file.

Job Output Directory

If the user does not specify a Nitro job directory (using `--job-dir` command line option), Nitro defaults to the `$HOME/nitro/<job id>` directory.

This directory contains the Nitro job and task log files, Nitro diagnostic log files, and temporary files Nitro uses to recover a canceled job. Users' home directories on these hosts should be mapped to a shared file location accessible to the users either from the system's login hosts, or outside of the cluster. Nitro will need to create directories and files in the Nitro job directory.

Recommended NFS Settings

If the job directory and task file are located on an NFS, then depending on the NFS cache settings, system administrators and users should be aware of the following:

- Job Directory - Users may not see job progress for several seconds after Nitro writes the files.
- Task File - If using Nitro in *linger mode*, Nitro may not recognize that new tasks have been added to the task file for up to 30 seconds.
- System Configuration (caching/tuning) - Add "lookupcache=none" to the NFS file system mount options in /etc/fstab for all the hosts using the NFS to prevent caching delays.

Related Topics

- [Resource Manager Launch Scripts on page 80](#)
- [Command Line Flags, Options, and Positional Parameters on page 82](#)
- [Linger Mode on page 77](#)

Run Nitro Without a Scheduler

Nitro works well running as a job invoked via a scheduler and/or resource manager.

However, if you have a set of hosts that are dedicated to high-throughput computing and would like to run Nitro jobs on these machines, or have Nitro always waiting (ready to run tasks at any time), you can run Nitro without the use of a scheduler.

This topic explains how to run Nitro without the use of a scheduler (also referred to Nitro standalone).

In this topic:

- [Selecting a Job ID on page 59](#)
- [Starting Workers on page 60](#)
- [Starting the Coordinator on page 60](#)
- [\(Optional\) Changing the Job Directory on page 61](#)

Selecting a Job ID

Since you do not have a scheduler to supply a job ID, Nitro will create a default job ID based on the current date and time in the format "YYMMDDHHMMSS". Since the workers and coordinator will be started separately, it is recommended that you designate a job ID and set the "--job-id <job id>" command line option on the worker and coordinator command lines with that job ID. Setting the job ID provides consistency between the workers and the coordinator when referencing the job; for example, when viewing job output and log files.

Starting Workers

You must start a worker on each host you want to execute Nitro tasks by starting it manually on a terminal on that host, or by running a remote command through ssh.

A worker command line must include an "--coord" argument that lists the host name and port number (if other than default port "47000") of the coordinator.

When each worker connects to the coordinator, it identifies itself using the --name option. If --name is not supplied, the worker's identification is the worker node's name.

i The name supplied to the coordinator via --workers or --workers-file must match the --name argument specified on the worker's command line.

For example, if you have the coordinator running on host "node001" and a worker running on host "node002", then the command line to start the worker would look like the following:

```
/opt/nitro/bin/nitro --mode=worker --job-id MyJob01 --coord node001 --name node002
```

Alternately, to run the worker using ssh (assuming ssh keys have been exchanged between the coordinator and worker), it would look like the following:

```
ssh user@node002 '/opt/nitro/bin/nitro --mode=worker --job-id MyJob01 --coord node001 --name node002'
```

Starting the Coordinator

The coordinator requires either a list of worker names or a session key that workers will use to attach to the coordinator. When supplying a list of worker names, these workers will be the only workers authorized to connect to the coordinator and receive workload assignments. If you specify a session key, any worker with the session key will be able to attach and receive workload.

Using Worker Names

A list of worker names can be specified on the command line, with worker names separated by the plus symbol (+). For example:

```
/opt/nitro/bin/nitro --mode=coord --job-id MyJob01 --workers node002+node003+node004+node005 ~/taskfile.txt
```

Alternately if it is a large list, it may be written to a file with one name per line, where the name is what the worker identifies itself as to the coordinator. For example:

```
echo -e "node001\nnode002\nnode003\nnode004\nnode005" > nodelist.txt
/opt/nitro/bin/nitro mode=coord --job-id MyJob01 --workers-file nodelist.txt
~/taskfile.txt
```

host node001 is the coordinator and hosts node002-node005 are workers.

Using a Session Key

To start the coordinator with a session key, use the --key <keyvalue> command line option instead of the --workers or --workers-file command line options. Any worker that is started with the --key command line option with the correct <keyvalue> will be able to attach to the coordinator. For example:

```
/opt/nitro/bin/nitro --mode=worker --coord node01 --job-id MyJob01 --key password1234  
/opt/nitro/bin/nitro --mode=coord --name=node01 --job-id MyJob01 --key password1234  
~/taskfile.txt
```

(Optional) Changing the Job Directory

The default job directory (the location to which Nitro will write the job and task log files) is `$HOME/nitro/<job id>`.

If you want to store results in a different directory, use the "--job-dir" command line option on both the coordinator and workers.

Related Topics

- [Command Line Flags, Options, and Positional Parameters on page 82](#)

Chapter 4 Using Nitro

This chapter provides information and instructions on using Nitro.

In this chapter:

- [Prepare a Nitro Job on page 63](#)
- [Submit a Nitro Job to a Scheduler on page 67](#)
- [Track Job Progress on page 70](#)
- [Dynamic Workload on page 76](#)

Prepare a Nitro Job

This topic provides information on the Nitro job's task file and performance tuning information.

In this topic:

- [Task File on page 63](#)
- [Performance Tuning on page 65](#)

Task File

The task file is a single file that contains a list of tasks to execute. Each line of the task file should contain only *one* task. You can add comments to your task file to help describe the tasks being performed, the data required, or other information that is pertinent to describing the tasks. Nitro also provides the capability to use task names and labels to help you organize your tasks.

Most of the tasks you create for a task file will probably run to completion fairly quickly, but it is possible that a task gets stuck in a loop or needs to run for a certain amount of time. Nitro by default limits tasks to 3,600 seconds (1 hour), but you can specify the limit to apply to the task by using the "maxtime" token. Time limits are specified in seconds. The following is an example of a task definition that limits a task to 30 seconds.

```
name=S23T01 maxtime=30 cmd=/opt/framemaker/bin/framegen -i /shared/scene23.def -tindex
1
```

Tasks

A task line can be as simple as the command you want to execute. For example, if you want to run a program called "framegen", input a file from a shared directory, and process the frame starting at time index "0" (zero), the command line might look like as follows.

```
/opt/framemaker/bin/framegen -i /shared/scene23.def -tindex 0
```

Nitro uses name/value pairs before the command line that you want to execute to define Nitro-specific information, such as, specifying a task name, task labels (that you can use to categorize the task), maximum time a task will run, and the command to execute to run the tasks itself. The key words for these name/value pairs are:

```
"cores=<count>"
"env=<name=value>[, <name=value>, ...]"
"labels=<label>[, <label>, ...]"
"name=<task name>"
"maxtime=<time limit in seconds>"
"memory=<amount>"
"shell=[default | none | <shell path>]"
"cmd=<command line>"
```

i The optional name/value pairs must be prepended to the line containing the task command line. As soon as Nitro sees something that isn't a name/value pair, the task line parsing stops and the rest is assumed to be part of the command line to execute.

To make it clear where the task options end and your command line begins, include "cmd=" before your task's command line. This token is optional but helps to make the task definition easier to read when you are specifying other options. The following is an example command line with the "cmd=" token.

```
cmd=/opt/framemaker/bin/framegen -i /shared/scene23.def -tindex 0
```

Nitro organizes the tasks for tracking.

Nitro tracks tasks by a task ID and line number. Nitro automatically generates a task ID for each task definition in the task file. The first task definition receives task ID "1".

i Only a task definition will increment the task ID. Because a task file can have empty or comment lines, the task ID and the line number in the task file may not be the same for the task.

Nitro will create a report of all tasks run and will include the task ID and line number in this report. The task ID is passed to the task via the **\$NITROTASKID** environment variable.

To make Nitro tasks easier to track, or to search for specific tasks in the task completion report, add a unique task name to your task definition. Task names don't have to be unique, but creating a unique task name helps you identify specific tasks.

i You can use any naming scheme you want, as long as the name does not include spaces (which would indicate an end to the name/value pair).

For example, if you are processing data for scenes 21, 22, and 23, you can name the tasks according to scene and time index.

```
name=S21T00 cmd=/opt/framemaker/bin/framegen -i /shared/scene21.def -tindex 0
name=S21T01 cmd=/opt/framemaker/bin/framegen -i /shared/scene21.def -tindex 1
name=S22T00 cmd=/opt/framemaker/bin/framegen -i /shared/scene22.def -tindex 0
name=S22T01 cmd=/opt/framemaker/bin/framegen -i /shared/scene22.def -tindex 1
name=S23T00 cmd=/opt/framemaker/bin/framegen -i /shared/scene23.def -tindex 0
name=S23T01 cmd=/opt/framemaker/bin/framegen -i /shared/scene23.def -tindex 1
```

Nitro makes the task name available to the task via the **\$NITROTASKNAME** environment variable when it executes the task. If the task command line includes the environment variable, it is substituted by its value before the command executes.

You can also use task labels to organize or identify the tasks. You can use multiple labels to describe a task. Multiple label values are separated by a comma between them; spaces are not allowed.

For example, if scene 22 contains a green screen that needs additional processing after this job completes, you can include the label "green" on all of the tasks for this scene.

```
name=S21T00 cmd=/opt/framemaker/bin/framegen -i /shared/scene21.def -tindex 0
name=S21T01 cmd=/opt/framemaker/bin/framegen -i /shared/scene21.def -tindex 1
name=S22T00 labels=green /opt/framemaker/bin/framegen -i /shared/scene22.def -tindex 0
name=S22T01 labels=green /opt/framemaker/bin/framegen -i /shared/scene22.def -tindex 1
name=S23T00 cmd=/opt/framemaker/bin/framegen -i /shared/scene23.def -tindex 0
name=S23T01 cmd=/opt/framemaker/bin/framegen -i /shared/scene23.def -tindex 1
```

Performance Tuning

Assignment Size

Each set of tasks that a coordinator sends to a worker is called an assignment. Nitro is most efficient when it can send a large enough assignment to each worker to keep the worker busy for at least 10 seconds before requesting more work. On the other hand, if you have a heterogeneous set of hosts with a wide variance in performance characteristics, you don't want one worker taking a very long time completing its assignment after all of the other workers have finished.

i You can specify an assignment size as small as "1" (which could be useful for tasks that need to use all of the available OS cores) and as large as "1000" (useful to keep worker cores busy with tasks of extremely short duration).

i Assignment sizes don't need to be evenly divisible by the number of OS cores available. Nitro will try to send the worker a second assignment when the worker gets about half way done with the current assignment so the second assignment will start running tasks as soon as an OS core becomes idle from the previous assignment.

Nitro uses a default assignment size of 250 tasks. If this is too many or too few tasks per assignment for your configuration, you can change it by using the `--assignment-size` command line option for the coordinator.

For example, if your nodes are all running 16 OS cores and each task takes 2 seconds to complete, each assignment of default size will take 31.25 seconds to complete (250 tasks at 2 seconds each divided by 16 OS cores), so you might want to change the assignment size to "80" to get an assignment time of closer to 10 seconds with the command line option.

i Adaptive Computing recommends a 10-20 second assignment duration to optimize node utilization and to prevent "tailing" jobs (job where at its end there is only one or a few workers executing a large assignment and other workers are idle).

```
--assignment-size 80
```

If submitting a job to a job scheduler, you can change the assignment size by setting the `NITRO_COORD_OPTIONS` environment variable so it contains the `--assignment-size` command line option.

Thread Control

Nitro typically runs one task per available OS core on each worker. However, you can configure Nitro to run more tasks than OS cores (over-subscription), fewer tasks than OS cores (under-subscription) or a specific number of OS cores. You might want to over-subscribe the available OS cores if you are not utilizing the full capacity of the host. You may need to under-subscribe cores if

background tasks are running on the nodes. To over- or under-subscribe, use the `--thread-ratio` command line option.

```
--thread-ratio <ratio>
```

<ratio> only applies to worker nodes. However, if you are using the `--run-local-worker` command line option, then the thread ratio will be passed on to the coordinator's local worker.

`--thread-ratio` also lets you specify over- or under-subscription properly in a heterogeneous host environment where hosts have different numbers of processors, cores, or threads. For example, if your hosts are all single socket, oct-core with hyper-threading enabled (16 total OS cores), but you want to over-subscribe by a factor of 1.5x, you could accomplish this by adding "`--thread-ratio 1.5`" to the worker command line to give each worker the ability to run 24 concurrent tasks. Alternately, if your tasks are all designed to use 2 OS cores each (multi-threaded application), you could use "`--thread-ratio 0.5`".

There may also be cases where you want to specify the *exact* number of OS cores to be used by the worker, such as when you have tasks that will use all available OS cores. In that case, you would use the `--thread-count` command line option to specify a thread count of 1 (`--thread-count 1`).

Run a Worker on the Coordinator Node

In configurations where you will be running less than 20 worker hosts, the coordinator node may be underutilized. To remedy this situation you may want to run a worker on the coordinator host so you can use its resources more effectively. To run a worker on the coordinator host, include the `--run-local-worker` flag on the coordinator's command line *or* you can explicitly start a worker Nitro process on the host.

i Nitro will calculate the number of threads that the local worker should run so the coordinator is not starved for CPU cycles; causing it to slow down all the other workers.

Task Execution Environment Variables

Nitro will pass several environment variables to your tasks when it executes them.

Valid environment variables:

- `$NITROJOBID` – Job ID of the Nitro job.
- `$NITROJOBDIR` – Job directory to which Nitro writes log files. This directory can be used to store output files from your tasks.
- `$NITROTASKCORES` – Number of cores allocated to the task.
- `$NITROTASKID` – Task ID of the task. The task ID is a number that starts at 1 and increments by 1 for each task definition (valid or invalid) in the task file. Commented and empty lines are not counted; if the task file contains such, the task ID and the line number will diverge.
- `$NITROTASKMEMORY` – Amount of memory (in MB) allocated to the task.
- `$NITROTASKNAME` – Task name, if provided by the task definition.
- `$NITROTASKTIME` – Task time limit, specified by "maxtime" in the task definition.
- `$NITRO_TASK_FILE` - Can be used with normal file names that do not use spaces, but **MUST** be used if the user submits more than one task file.

- `$NITRO_LONG_TASK_FILE` - Can be used with normal file names that do contain spaces, but **MUST** be used if the file name contains spaces. This variable can only contain one file name. You cannot submit multiple file names containing spaces.



Nitro reads a portion of the task file at a time. While the Nitro job is running, do *not* add or remove any task definitions or comment lines in the task file. Changes to the task file could cause line numbers to be changed and jobs to not run or be accidentally rerun.

Related Topics

- [Command Line Flags, Options, and Positional Parameters on page 82](#)

Submit a Nitro Job to a Scheduler

This topic provides information about submitting a Nitro job and executing Nitro. The user creates the user job script and the system administrator sets up and configures the Nitro launch script, which the user job script executes to run Nitro.

Adaptive Computing's objective for Nitro is two-fold; first, allow the user to use the same job script for a Nitro job regardless of the scheduler used by a system to schedule jobs and, second, allow the administrator to customize Nitro for the system based on the scheduler and/or resource manager the system uses. This means the user job script can be scheduler-agnostic while the Nitro launch script (executed by the user job script) must be scheduler- or resource manager-specific.

In this topic:

- [User Job Script on page 67](#)
- [Nitro Launch Script on page 67](#)
- [Customize the Nitro Launch Script on page 68](#)
- [Launch Script Environment Variables on page 68](#)
- [Moab/Torque Customization Commands and Options on page 68](#)

User Job Script

Typically, the system administrator provides a sample Nitro job script to users to customize as needed.



A sample user job script is located in the Nitro installation's "samples" subdirectory.

Users can set launch script environment variables at the user job script-level to affect Nitro's behavior. See [Launch Script Environment Variables on page 68](#) for a list of available variables.

The user job script also contains the information to execute the Nitro launch script (`/opt/nitro/bin/launch_nitro.sh`).

Nitro Launch Script

Adaptive Computing provides a configured Nitro launch script for different schedulers or resource managers. You can also build a new script. See [Resource Manager Launch Scripts on page 80](#).

During the installation of Nitro, the system administrator copies the Nitro launch script (`/opt/nitro/scripts` directory) to the directory (`/opt/nitro/bin`) from where the user job script executes it.

Customize the Nitro Launch Script

The system administrator may customize the Nitro launch script to suit the needs of the system. The Nitro launch script may examine and modify, or simply pass through, the command line options specified by the environment variables set by the user job script. The command line options are then passed via the environment variables to the Nitro workers and coordinator that are started by the Nitro launch script.

Launch Script Environment Variables

The Nitro launch script looks for several environment variables. Users can also customize these variables in the user job script.

- `NITRO_TASK_FILE`
- `NITROJOBID`
- `NITRO_OPTIONS`
- `NITRO_COORD_OPTIONS`
- `NITRO_WORKER_OPTIONS`.

NITRO_TASK_FILE

The `NITRO_TASK_FILE` environment variable must contain the path to the task file, and the task file must be accessible to the coordinator host that will run the job.

Users should put the task file in a shared location accessible by any host in the system.

NITROJOBID

The `NITROJOBID` environment variable is created automatically by the launch script, based on the resource manager's job ID, if the user job script does not provide one.

Unless you are restarting a job that partially completed and was canceled, you don't need to set this environment variable. If you specify this environment variable in the job's submission, it will override the resource manager job ID and Nitro will use the value you supplied.

i If your job scheduler and resource manager use different numbering systems, the job ID that Nitro will use is the one that it gets from the resource manager. You may want to submit the job directly to the resource manager in this case to avoid confusion. Check with your system administrator to find out if your job scheduler's and resource manager's job ids are synchronized.

NITRO_OPTIONS, NITRO_COORD_OPTIONS, or _NITRO_WORKER_OPTIONS

Any command line options the user job script needs to pass to Nitro must be contained in either the `NITRO_OPTIONS`, `NITRO_COORD_OPTIONS`, or `NITRO_WORKER_OPTIONS` environment variables.

Moab/Torque Customization Commands and Options

If using Moab/Torque, be aware of the following:

- Nitro will consume all available OS cores on the hosts on which it runs.
 - If Nitro is running exclusively on hosts, you may want to configure your system to run a single job per host (for example, in Moab, set NODEACCESSPOLICY to SINGLEJOB in the moab.cfg file).
 - If you will run other workload on the system that may run multiple jobs per host, you will need to provide a script or instruct users on how to specify their Nitro jobs to run with a single task per host configuration. In Moab and Torque the jobs should be submitted (using either msub or qsub) with the "-l nodes=<node count>:tpn=1" command line option.

Examples

The sample launch_nitro.sh job scripts provided in /opt/nitro/scripts/<resource manager>/ all expect environment variables. You can set environment variables for the Nitro launch script at job submission or define them in the user job script.

Example: Setting Environment Variables at Job Submission

The "-v" option in Moab's msub command sets environment variables before calling the user job script. The user job script inherits these environment variables.

i The user job script can add to or overwrite any environment variable defined in msub's "-v" option.

- User job script (/home/jdoe/user_job_script.sh)

This script must exist on the host where you execute the msub command. In this example, this user job script exists in jdoe's home directory.

```
exec /opt/nitro/bin/launch_nitro.sh
```

- msub example

```
msub -lnodes=5 -ltpn=1 -lwalltime=600 -v "NITRO_TASK_FILE=/home/jdoe/nitro/monte_sim.txt,NITRO_OPTIONS=--job-dir /home/jdoe/monte01,NITRO_COORD_OPTIONS=--run-local-worker" /home/jdoe/user_job_script.sh
```

- The task file monte_sim.txt contains the list of task command lines that you want to execute.
- The "--job-dir" sets the path to which Nitro will write the job and task logs.
- The last parameter is the path of the user job script.

Example: Setting Environment Variables In the User Job Script

- User job script (/home/jdoe/user_job_script.sh)

This script must exist on the host where you execute the msub command. In this example, this user job script exists in jdoe's home directory.

```
NITRO_TASK_FILE=/home/jdoe/nitro/monte_sim.txt
NITRO_OPTIONS="--job-dir /home/jdoe/monte01"
NITRO_COORD_OPTIONS="--run-local-worker
exec /opt/nitro/bin/launch_nitro.sh
```

- **msub example**

```
msub -lnodes=5 -ltpn=1 -lwalltime=600 /home/jdoe/user_job_script.sh
```

- The task file `monte_sim.txt` contains the list of task command lines that you want to execute.
- The `--job-dir` sets the path to which Nitro will write the job and task logs.
- The last parameter is the path of the user job script.
- `NITRO_TASK_FILE`, `NITRO_OPTIONS` and `NITRO_COORD_OPTIONS` are set in the user job script.

Track Job Progress

This topic provides information on viewing job progress and output.

In this topic:

- [Introduction on How Nitro Tracks the Job on page 70](#)
- [Job Log on page 71](#)
- [Task Log on page 74](#)

Introduction on How Nitro Tracks the Job

Nitro will print some job information to stdout, such as what workers attached, how many tasks have been run, if any tasks failed, etc.

If your Nitro job is submitted through a scheduler, you may not see any of this until the job has completed and the resource manager has copied the job output to your job's submission directory.

However, Nitro provides a tool called `nitrostat` to display status information while the job is running. `nitrostat` is located in the `nitro/bin` directory where Nitro was installed.

Nitro creates two files that you can use to stay up-to-date on the progress of your job.

- `nitro_<jobid>.joblog.txt` - Information about the job in general.
- `nitro_<jobid>.tasklog.txt` - Listing of individual tasks that have completed along with performance statistics collected from running the task (duration and memory usage) and the task output to stdout and/or stderr.

i Both files are written to the job directory that you provide using the `--job-dir` command line option when submitting your job, or to the default job directory `$HOME/nitro/<jobid>`.

Job Log

To see job status using `nitrostat`, you will need the job ID. The job ID is the job ID reported to you when you submitted the job to the scheduler *or* that you set manually via the `--job-id` command line option in the `NITRO_OPTIONS` environment variable or via the `NITROJOBID` environment variable.

- The default location for the job and task logs are in your "`$HOME/nitro/<jobID>`" directory.
- You can also use the "`--job-dir`" command line option to specify a different job directory if you are not using the default location.

Nitro Job Progress Report

The Nitro job progress reports lets you see the current contents of a job log file.

For example, let's say you have a job that was run by your resource manager as job "23576", running "`/opt/nitro/bin/nitrostat 23576`" shows you the job's progress.

```
Nitro Job Progress Report

Start Time   : 2016-02-10 09:10:11-0600
Current Time : 2016-02-10 09:10:42-0600
Elapsed Time : 31 seconds (00:00:31)

Job Id       : 23576
Coordinator  : node01
Task Log     : /home/jdoe/jobs/23576/nitro_23576.tasklog.txt
Task File    : /home/jdoe/jobs/survey03.tasks
  File Size  : 123366
  Est Tasks  : 3016
  Processed  : 75%

Tasks
-----
Pending      : 500
In Progress  : 500
Completed    : 1250
  Success    : 1250
  Failure    : 0
  InsufRes   : 0
  Timeout    : 0
  Invalid    : 0
  Tasks/sec  : 40.3
Total Tasks  : 2250

Workers
-----
Host  Pid  Thrds Status  Assigned Running Completed  Success  Failure  InsufRes
Timeout Tasks/sec  AsgmtDur
node02 6851   12 running   1250    250    1000    1000     0     0
0      36.0    8.0
node03 14988   4 running    500    250    250    250     0     0
0      9.3    27.0
```

The following describes the fields and their output descriptions.

- **Start Time** – Date and time the coordinator started running.
- **Current Time** – Current date and time the report was generated (reports are generated every 5 seconds).

- Elapsed Time – Amount of time the coordinator has been working on the tasks.
- Job Id – Job ID that Nitro was passed on its command line. Typically assigned by the resource manager, but can be assigned by the user.
- Coordinator – Host name on which the coordinator is running.
- Task Log – Path and file name of the task log file that is generated by the coordinator.
- Task File – Path and file name of the task file.
- File Size – File size of the task file.
- Est Tasks – Number of tasks the coordinator estimates in the task file. Since Nitro doesn't read the entire task file on startup, an estimate is given based on lines read from the file so far.
- Processed– Percentage of the task file that has been read by the coordinator.
- Tasks Section: Lists the counts of tasks in each category
 - Pending – Number of tasks that have been put into assignments and are waiting to be sent to a worker.
 - In Progress– Number of tasks in assignments sent to the workers for which workers have not yet returned results.
 - Completed – Number of tasks in assignments that have been completed (workers have returned results).
 - Success – Number of completed tasks that were successful (the task returned an exit code of 0).
 - Failure – Number of completed tasks that returned an exit code other than 0.
 - InsufRes – Number of tasks that could not be run because the requested resources for the task was not available.
 - Timeout – Number of completed tasks that ran longer than the task "maxtime" parameter and were terminated by the worker.
 - Invalid – Number of task definitions that contained errors and could not be run.
 - Tasks/sec – Number of tasks per second based on the time that the coordinator sends the first assignment until the time the report is generated. If in *linger mode*, this will only be calculated for the last 60 seconds.
 - Total Tasks – Total number of tasks including completed and invalid tasks.
- Workers Section: List by worker
 - Host – Host name and port (if not the default port) of the worker.
 - Pid – Process ID of the worker.
 - Thrds –Number of task launch threads the worker is using to run tasks.
 - Status – Status of the worker. This may be "unconnected", "running", "unresponsive", "closing", or "closed".

- Assigned – Number of tasks assigned to this worker so far.
- Running – Number of tasks in assignments currently allocated to the worker.
- Completed – Number of tasks in assignments the worker has completed.
- Success – Number of successfully completed tasks.
- Failure – Number of tasks that returned an exit code other than 0.
- InsufRes – Number of tasks that could not be run because the requested resources for the task was not available.
- Timeout – Number of tasks that exceeded the tasks "maxtime" threshold and were terminated by the worker.
- Tasks/sec – Number of tasks per second that the worker has completed so far. In *linger mode* this is only calculated for the last 60 seconds.
- AsgmtDur – Average assignment duration in seconds.

Job Completed Report

Once the job has completed, the job report will show "(final)" on the end of the first line of the report and Current Time is replaced with Finish Time (after Start Time). The following example is based on the previous example for job "23576" .

```

Nitro Job Progress Report (final)

Start Time   : 2016-02-10 09:10:11-0600
Finish Time  : 2016-02-10 09:11:36-0600
Elapsed Time : 85 seconds (00:01:25)

Job Id       : 23576
Task Log     : /home/jdoe/jobs/23576/nitro_23576.tasklog.txt
Task File    : /home/jdoe/jobs/survey03.tasks

Tasks
-----
Pending      : 0
Running      : 0
Completed    : 3000
  Success    : 3000
  Failure    : 0
  InsufRes   : 0
  Timeout    : 0
  Invalid    : 0
  Tasks/sec  : 35.3
Total Tasks  : 3000

Coordinator
-----
Host         : node01
Threads     : 8

Worker Resources
-----
Workers      : 2
Threads     : 16

Workers
-----
Host  Pid   Thrds Status Assigned Running Completed Success Failure InsufRes
Timeout Tasks/sec AsgmtDur
node02 6851   12  closed    2250      0     2250     2250      0      0
0      29.2   8.3
node03 14988  4  closed     750      0      750      750      0      0
0      8.8    35.7

```

Task Log

The task log file contains a listing of all tasks that have been completed and some statistics about the tasks duration and memory consumption. This file is named `nitro_<JobID>.tasklog.txt` and is located in the same directory as the job log file.

The task log file is tab-delimited, so you can easily import it into a spreadsheet or database, or process it using another program. You can also view the task log using the `nitrostat` utility.

JobID	TaskID	Line	Name	Status	ExitCode	Hostname	StartTime
		Duration	UserCPU	SystemCPU	VirtualMem	PhysicalMem	Labels
Output							
foo	1	1	task001	Success	0	localhost:10004	2015-06-18_
		15:26:52.954-0600	1.005	0.000	0.000	7364608	630784
foo, foobar, foobaz, xyz							
foo	2	2	task002	Success	0	localhost:10004	2015-06-18_
		15:26:52.954-0600	1.007	0.000	0.000	87834368	630784 foo, foobar, xyz
foo	3	3	task003	Success	0	localhost:10004	2015-06-18_
		15:26:52.954-0600	1.005	0.000	0.000	71728640	901120 foo, xyz
foo	4	4	task004	Success	0	localhost:10004	2015-06-18_
		15:26:52.955-0600	1.005	0.000	0.000	38837504	630784
foo, foobar, foobaz, abc							
foo	5	5	task005	Success	0	localhost:10004	2015-06-18_
		15:26:53.960-0600	1.004	0.000	0.000	405946368	630784 foo, foobar, abc
foo	6	6	task006	Success	0	localhost:10004	2015-06-18_
		15:26:53.961-0600	1.005	0.000	0.000	405946368	946176 foo, abc
foo	7	7	task007	Success	0	localhost:10004	2015-06-18_
		15:26:53.961-0600	1.003	0.000	0.000	405946368	630784
foo	8	8	task008	Success	0	localhost:10004	2015-06-18_
		15:26:53.966-0600	1.003	0.000	0.000	405946368	700416
foo	9	9	task009	Success	0	localhost:10004	2015-06-18_
		15:26:54.965-0600	1.005	0.000	0.000	405946368	630784
foo	10	10	task010	Success	0	localhost:10004	2015-06-18_
		15:26:54.965-0600	1.003	0.000	0.000	405946368	630784
foo	11	11	task011	Success	0	localhost:10004	2015-06-18_
		15:26:55.973-0600	1.005	0.000	0.000	7364608	630784
foo	12	12		Success	0	localhost:10004	2015-06-18_
		15:26:55.973-0600	1.004	0.000	0.000	405946368	626688
foo	13	14	fail	Failure	1	localhost:10004	2015-06-18_
		15:26:55.973-0600	0.005	0.000	0.000	8192	4096
foo	14	16	stderr	Success	0	localhost:10004	2015-06-18_
		15:26:55.974-0600	0.005	0.000	0.000	405946368	536576
foo	15	18	stderr_fail	Failure	1	localhost:10004	2015-06-18_
		15:26:55.979-0600	0.005	0.000	0.000	405946368	1228800
ERROR MESSAGE							
foo	16	20	overtime	Timeout	-9	localhost:10004	2015-06-18_
		15:26:55.980-0600	2.006	0.000	0.000	405946368	970752
maxtime exceeded, process was killed							
foo	17	21		Success	0	localhost:10004	2015-06-18_
		15:26:55.985-0600	1.002	0.000	0.000	405946368	626688
foo	19	23		Success	0	localhost:10004	2015-06-18_
		15:26:56.979-0600	1.007	0.000	0.000	405946368	970752
foo	20	24		Success	0	localhost:10004	2015-06-18_
		15:26:56.988-0600	1.003	0.000	0.000	405946368	724992
foo	21	25		Success	0	localhost:10004	2015-06-18_
		15:26:57.986-0600	1.005	0.000	0.000	405946368	724992
foo	22	26		Success	0	localhost:10004	2015-06-18_
		15:26:57.988-0600	1.005	0.000	0.000	405946368	970752
foo	23	27		Success	0	localhost:10004	2015-06-18_
		15:26:57.988-0600	1.005	0.000	0.000	405946368	630784
foo	24	28		Success	0	localhost:10004	2015-06-18_
		15:26:57.995-0600	1.005	0.000	0.000	405946368	630784
foo	25	29		Success	0	localhost:10004	2015-06-18_
		15:26:58.993-0600	1.005	0.000	0.000	405946368	974848
foo	26	30		Success	0	localhost:10004	2015-06-18_
		15:26:58.994-0600	1.004	0.000	0.000	405946368	626688

The task log contains the following fields.

- JobID – Job ID that was passed to Nitro using the "--job-id" command line option.
- TaskID – Task number within the Nitro job.
- Line – Line number in the task file of the task definition.

- Name – Task name supplied in the task definition by the "name=<name>" option.
- Status – One of "Success", "Failure", "InsufRes", "Timeout", or "Invalid".
- ExitCode – Numerical exit code returned by the task.
- Hostname – Name of the worker that executed the task.
- StartTime – Date and time the worker actually started the task.
- Duration – Number of seconds the task ran (millisecond resolution).
- UserCPU – Number of seconds the task ran in user mode (millisecond resolution).
- SystemCPU – Number of seconds the task run system calls (millisecond resolution).
- VirtualMem – Maximum virtual memory allocated to the task in bytes.

i The operating system may allocate shared memory and may charge a proportion of this shared memory to random tasks.

- PhysicalMem – Maximum physical memory allocated to the task in bytes.
- Labels – Optional task labels specified by the task definition.
- Output – stdout and/or stderr. If a task outputs to both stdout and stderr, both are displayed in the format <stdout>/<stderr>.

Related Topics

- [Command Line Flags, Options, and Positional Parameters on page 82](#)
- [nitrostat on page 89](#)

Dynamic Workload

This topic identifies activities pertaining to dynamic workload.

Nitro jobs are flexible in the number of resources they can use to accomplish the tasks given them. Worker nodes can be added to a job or taken away without any adverse consequences (other than the job running more slowly). You can also add workload by appending the task file.

In this topic:

- [Removing Worker Nodes on page 76](#)
- [Adding Worker Nodes to a Running Job on page 77](#)
- [Linger Mode on page 77](#)

Removing Worker Nodes

If nodes are needed for a more important task, the workers can be killed, and their assignments will be returned to the coordinator.

i Killing the worker with a SIGTERM signal will allow the worker to send a partial assignment completion report to the coordinator. Be aware that if a worker is killed, the tasks that are running, may be run again when the assignment is given to a different worker to complete. Therefore, it is important to program your tasks to exit if the work has already been completed or overwrite the previous result.

Adding Worker Nodes to a Running Job

While the workload is being executed, workers can be added to the coordinator. The coordinator requires either a list of worker names or a session key that workers will use to attach to the coordinator to receive workload assignments. If you specify a list of worker names, only those workers will be authorized to connect to the coordinator. If you specify a session key, any worker with the session key will be able to connect to the coordinator.

Once the coordinator exits, the job is finished and workers won't be able to connect.

Linger Mode

If you need to keep a coordinator up continually to respond to workload that could be added at any time, you can use the "--linger" command line option on the workers and coordinator to allow Nitro to stay resident and not exit when the tasks are completed.

Nitro provides a message-based process to dynamically add workload to Nitro. Contact Adaptive Computing Professional Services for more information on dynamically adding workload.

Chapter 5 References

This chapter provides additional information for system administrators and users.

In this chapter:

- [Nitro Configuration File on page 79](#)
- [Resource Manager Launch Scripts on page 80](#)
- [Command Line Flags, Options, and Positional Parameters on page 82](#)
- [Task File on page 86](#)
- [nitrostat on page 89](#)
- [Job Recovery on page 91](#)
- [Coordinator Resiliency on page 92](#)
- [Glossary on page 92](#)

Nitro Configuration File

The `nitro.cfg` file is available to system administrators.

Nitro looks for this configuration file in the `/opt/nitro/etc` directory, which must be a peer to the `/opt/nitro/bin` directory. If found, Nitro will load configuration options specified in the `nitro.cfg` file.

`nitro.cfg` Configuration Options

These configuration options are available to customize the `nitro.cfg` file:

- `assignment-size <size>` – Sets the default assignment size.
- `coord-threads <count>` – Indicates to the coordinator how many threads to reserve for the coordinator when allocating cores to a local worker (when using "`--run-local-worker`" on the coordinator command line). Default is 2. Adaptive Computing recommends setting the `<count>` value to 1 if all jobs will use less than 20 nodes and setting the `<count>` value to 4 if the jobs require a large number of nodes (greater than 50) to run.

i There is also a "`--coord-threads`" command line option, that if set, overrides the `<count>` specified here (`nitro.cfg` file).

- `default-shell <shell path>` – Allows the configuration of the default shell used by Nitro to launch tasks. The default value is `"/bin/bash"`. In high throughput usages where many very small tasks need to be launched as quickly as possible, it may be beneficial to use a more compact shell such as the Bourne shell or Korn shell. To set this value, specify the fully qualified path to the shell such as `"/bin/sh"`.

- `default-shell-command <command>` – Allows configuration of the command line parameter to the shell that Nitro uses when launching a task. The default shell command is `-c` so that Nitro will execute `"/bin/bash -c <task command line>".` Set this value if you need to customize the launch command or are using a shell that uses a different command line option to launch a command line.
- `disable-affinity` – Instructs the workers that they should not track and set the task's affinity.

i There is also a `--disable-affinity` command line option, that if set, overrides the setting here (nitro.cfg file).

- `max-cpu-threshold <value>` – Causes Nitro to reduce task threads when the threshold is reached. The `<value>` is a percentage of system load, where 100% is equivalent to the number of cores in the node and is measured by the 60 second load average.
- `maxtime-limit <period>` – Lets you set the maxtime option's upper limit for all Nitro job task definitions. If this option is not specified, the coordinator will use its default of 1 day (86400 seconds). This value can be specified as a number of seconds, or by using the Days:Hours:Minutes:Seconds format (ie: 7:0:0:0 = 7 days).
- `min-memory-threshold <threshold MB>` – If set to a value other than 0 (zero), causes Nitro to check the workers available physical memory before starting tasks. If the available physical memory drops below this value, Nitro will stop running tasks until available memory rises above the threshold. This can be used to throttle tasks or account for background activity that is consuming memory and causing the system to swap memory to disk. It is important to select a threshold that will allow the system to throttle before swapping starts to happen. 2GB (2000MB) is a good threshold to start with.
- `throttle-period <period>` – Sets the amount of time (in seconds) that Nitro will wait before checking for throttling conditions (such as the memory threshold). Nitro defaults to checking every 2 seconds. Setting this to 0 (zero) causes Nitro to check when tasks complete, or 1 second whichever comes first.
- `task-output-limit <value>` – Sets the number of bytes that will be captured by stdout and stderr (each) and written to the task log file. The default value is 512 bytes. Nitro captures the stdout and stderr strings output by tasks, but only retains the last n characters as set by the limit to reduce communication and storage overhead.

Resource Manager Launch Scripts

Launch scripts have been provided for Torque, SLURM, LSF, and Cray resource managers or environments. The launch scripts can be found in the `/opt/nitro/scripts` directory. If you use another resource manager, you may need to build a new script. Also, if you want to change the basic configuration or paths used with your installation, you may want to customize the launch script that the users of your system will use to start Nitro.

There are several basic points that a launch script needs to cover to interface the resource manager with Nitro.

1. Getting the resource manager job ID and passing it to Nitro. See [Resource Manager Job ID on page 81](#).
2. Specifying the location of the Nitro binary. See [Location of the Nitro Binary on page 81](#).
3. Getting the list of nodes that Nitro is to run on for the current job. See [List of Job Nodes on page 81](#).
4. Launching the Nitro workers and coordinator. [Launch Nitro Workers and Coordinator on page 82](#).
5. Customize the command line parameters of workers and/or coordinator. See [Customize Command Line Parameters on page 82](#).

Resource Manager Job ID

Nitro uses the job ID to customize the output files so if several copies of Nitro are running at the same time they don't corrupt each other's information. Torque, for example, defines the `$PBS_JOBID` environment variable that contains a job ID as defined by Torque.

The provided launch scripts add the "`--job-id <job id>`" parameter to Nitro's command line parameters (to the workers and the coordinator) if a job ID is provided, or if no job ID is found, then it defaults to a job ID with the format "YYYYMMDDHHMMSS" containing the date and time the Nitro launch script runs.

Location of the Nitro Binary

The launch script assumes the Nitro binary will be found in the `/opt/nitro/bin` directory. This requires that Nitro has been installed to each node, or that the `/opt/nitro` directory has been mapped to a remote file system.

If you have installed Nitro to a directory other than the default, you need to customize the Nitro launch script with this location. For example, if you installed Nitro to `/mysharedfs/nitro`, you need to change the line of the Nitro launch script

from

```
NITRO=/opt/nitro/bin/nitro
```

to

```
NITRO=/mysharedfs/nitro/bin/nitro
```

List of Job Nodes

Resource managers typically send the job to one node in the allocated set of nodes. The script then executes work on any of the remote nodes within the job allocation. The Nitro launch script then launches a Nitro worker on each of the remote nodes before launching the coordinator.

The coordinator only accepts a connection from the list of workers provided via the `--workers` or `--workers-file` parameter *or* by using the `--key` command line option to specify a session key "`<keyvalue>`".

Resource managers typically have an environment variable set with the list of nodes allocated to the job. For example, in Torque, this environment variable is "`$PBS_NODEFILE`" and contains the file that is accessible to the job containing the list of nodes allocated to the job.

The file containing the list of nodes is typically a file with a single node name per line, such as:

```
node01
node02
node03
```

Launch Nitro Workers and Coordinator

The Nitro launch script needs to remotely start up Nitro workers and return control to the script.

- Torque uses the "pbsdsh" command.
- SLURM uses the "srun" command.
- LSF uses the "blaunch" command.
- Cray systems use the "aprun" command.

i Please refer to your resource manager's documentation for instructions and options to run the remote command.

The workers are started first in the Nitro launch script so the coordinator can be executed last using "exec". This is so the coordinator gains control of the process. The Nitro launch script currently uses (and assumes) the first node in the nodes list as the coordinator and all other nodes as workers.

Customize Command Line Parameters

If all your user's jobs are expected to use less than 20 nodes, you may want to add the option to the script to run a local worker on the coordinator node to maximize task throughput. Users can already do this by adding the "--run-local-worker" flag to the NITRO_COORD_OPTIONS environment variable at job submission or in the user job script, but you may elect to add this to the Nitro launch script to reduce the parameters necessary for a user to configure. See [Submit a Nitro Job to a Scheduler on page 67](#).

If you want to add command line parameters to Nitro, the best way is to prepend the option to the beginning of either the NITRO_OPTIONS, NITRO_COORD_OPTIONS, or NITRO_WORKER_OPTIONS environment variable(s), as appropriate. For example, if you wanted to add the "--run-local-worker" flag to the coordinator command line, add the following line to the Nitro launch script after the line containing the NITRO_OPTIONS.

```
NITRO_OPTIONS="--job-id ${NITROJOBID} ${NITRO_OPTIONS}"
NITRO_COORD_OPTIONS="--run-local-worker ${NITRO_COORD_OPTIONS}"
```

Command Line Flags, Options, and Positional Parameters

This topic identifies the individual command line flags, options, and positional parameters recognized and/or required by Nitro.

In this topic:

- [Flags on page 83](#)
- [Options on page 83](#)
- [Positional Parameters on page 85](#)
- [Command Line Options per Nitro Mode on page 86](#)

Flags

- **Disable Affinity** – Instructs a worker that it should not track and set the task's affinity.

i This option overrides `--disable-affinity` in the `nitro.cfg` file.

```
--disable-affinity
```

- **Linger** – Tells Nitro to keep running after the initial tasks have completed. The `<timeout>` specifies the number of seconds that must pass after the last completed task file before Nitro closes (shuts down). A `<timeout>` value of `-1` indicates an indefinite period of time; Nitro will not close until a signal is given to close.

```
--linger <timeout>
```

- **Run Local Worker** – Runs a local worker on the coordinator's host.

```
--run-local-worker
```

- **Trust Workers** – Allows any worker to attach to Nitro and accept workload. Without this flag, the coordinator will only connect workers that were specified with the `--workers`, `--workers-file`, or `--key` command line option.

```
--trust-workers
```

Options

- **Mode** – Required command line parameter that indicates the role (`coord/worker`) of Nitro when started on a host.

```
--mode=[coord | worker]
```

- **Coordinator Host** – Specifies the coordinator to connect to when Nitro is running in worker mode (`--mode=worker`).

```
--coord <HOST[:PORT]>
```

- **Session Key** – Specifies a session key that can be used to authenticate workers to the coordinator. The session key must be provided on the workers and coordinator command lines. Any worker reporting in to the coordinator will be required to provide this key to be able to connect and receive workload. The session key is any string that does not contain spaces or any characters which the shell will interpret.

```
--key <keyvalue>
```

- **Key File** – File containing a passphrase that can be used to authenticate workers to a coordinator. If the file contains newline or tab characters, these will be removed from the passphrase.

```
--key-file <file>
```

- **Worker Hosts** – "+"-separated list of the host names of the worker hosts the job scheduler allocated to the Nitro job for its exclusive use.

```
--workers <hostlist>
```

- **Worker Hosts File** – File containing a list of the host names, one per line, the job scheduler allocated to the Nitro job for its exclusive use.

```
--workers-file <filename>
```

- **Name** – Name the worker uses to reference itself when communicating with the coordinator. For the coordinator, this is the name to which the workers will connect. If not specified, Nitro defaults to the node's name.

i The name supplied to the coordinator via `--workers` or `--workers-file` must match `--name`. If using `--trust-workers` or `--key`, any unique value can be used for `--name`. Workers' names must be unique within a coordinator group.

```
--name <name>
```

- **Port Base** – Base port number for port assignments (default 47000); used to override first port assignment by Nitro process. Valid range of ports is 47000-65535.

Each Nitro coordinator process requires four ports. A coordinator process will bind a range of four ports starting with the start port specified by this option as the starting point for port assignments.

```
-p <port>
--port1 <port>
```

- **Thread Count** – The quantity of threads the Nitro workers should use when executing tasks. This option is mutually-exclusive with the Thread Ratio option.

If this option and the Thread Ratio option are not given, a worker uses one task launch thread per OS core to which it is pinned.

The primary reason for this option is to explicitly specify a task-launch thread count for Nitro running a specific single application, usually on homogeneous hosts.

```
--thread-count <num>
```

- **Thread Ratio** – The ratio of task launch threads-to-OS "cores" the Nitro workers should use when creating task launch threads. This option is mutually exclusive with the Thread Count option.

If this option and the Thread Count option are not given, the ratio is "1.0", meaning a worker uses one task launch thread per OS core to which it is pinned.

Ratio is a positive real number (e.g., 1.5, 0.5, etc) that when multiplied with the count of OS cores to which a worker is pinned yields a count of the task launch threads it will use. The worker rounds the count to the nearest integer, with a minimum value of 1.

The primary reason for this option is to allow a user to over-subscribe or under-subscribe the task-launch thread count appropriately relative to the OS core count of heterogeneous hosts (e.g., 1.5 means 6 threads for a quad-core host and 24 threads for a 16-core host).

```
--thread-ratio <ratio>
```

- **Assignment Size** – The quantity of tasks the Nitro coordinator should pass to a Nitro worker at one time; default is 250.

```
--assignment-size <num>
```

- **Job Directory** – Specifies the path for the directory where Nitro will place its Job Progress Log and Completed Task Log files.

```
--job-dir <path>
```

- **Job ID** – Specifies the job ID for a specific Nitro run. The job ID may be used to create the job directory and certain file paths.

```
--job-id <jobID>
```

- **Coordinator Threads** – Indicates to the coordinator how many threads to reserve for the coordinator when allocating cores to a local worker (when using "--run-local-worker" on the coordinator command line). Default is 2. Adaptive Computing recommends setting the <count> value to 1 if all jobs will use less than 20 nodes and setting the <count> value to 4 if the jobs require a large number of nodes (greater than 50) to run.



This option overrides the --coord-threads setting in the nitro.cfg file.

```
--coord-threads <count>
```

- **Task Environment** – Specifies the environment variables to set in the task's execution environment. This is used by the worker but is also needed on the coordinator's command line if running a local worker. Multiple values can be specified by separating name/value pairs with a comma.

```
--task-env <ENVVARIABLE=value[,...]>
```

- **Debug Log File** – The path for the optional "debug log" file the Nitro coordinator can produce. If the path is not defined, the coordinator will use the default path `$NITROJOBDIR/nitro_$(NITROJOBID)-hostname_pid.log`.

```
--logfile <path>
```

- **Debug Log Level** – The level of debug log information the Nitro coordinator will output when tracing/logging. The default is "3" (information). The highest level allowed is "7" (debug).

```
--loglevel <num>
```

Positional Parameters

- **Task File Name** – Path of text file containing Nitro task definitions. This must be the last parameter on the coordinator's command line.

path

Command Line Options per Nitro Mode

The table that follows identifies which command line options Nitro uses in worker or coordinator mode. Some command line options are used in both modes and are listed in this table in the "Both" row.

Nitro Mode	Command Line Option
Coordinator	--assignment-size --coord-threads (if using --run-local-worker with the coordinator) --port1 --run-local-worker --trust-workers --workers --workers-file --key-file
Worker	--coord --disable-affinity (if <i>not</i> using --run-local-worker with the coordinator) --name --task-env (if <i>not</i> using --run-local-worker with the coordinator) --thread-count (if <i>not</i> using --run-local-worker with the coordinator) --thread-ratio (if <i>not</i> using --run-local-worker with the coordinator)
Both	--disable-affinity (if using --run-local-worker with the coordinator) --job-dir --job-id --key --linger --logfile --loglevel --mode --task-env (if using --run-local-worker with the coordinator) --thread-count (if using --run-local-worker with the coordinator) --thread-ratio (if using --run-local-worker with the coordinator)

A task file contains a list of Nitro task definitions (task execution options) along with the task command line Nitro will execute. Since the Nitro coordinator will be running on one of the nodes allocated to the Nitro job, the task file must be accessible to the node on which the coordinator will run.

The task file is a text file where each task definition must be contained on a single line. Lines of text may be terminated by either a Linux-style line ending (LF or '\n' new line character) or a Windows-style line ending (CR/LF - '\r\n' carriage return/line feed combination). The line number is reported in the task log so that errors in the task file can be quickly located and fixed.

The task file allows comment and empty lines. A hash symbol (#) in the first column of a line identifies a comment line.

Each task will be assigned a task ID, which will start at 1 and increment with each task line (comment and empty lines are not assigned a task ID). This task ID is passed to the task in the NITROTASKID environment variable.

Task Options

Task options are name/value pairs that are listed before the task's command line of the form "*<option>=<value>*". Task options must be specified *before* the task's command line to be executed. As Nitro parses the line, it will stop looking for name/value pairs as soon as it finds a character string that does not include the name/value delimiter (=) or is the "cmd" option. Everything after the "cmd=" option or the first string that is not delimited as a name/value pair will be considered part of the task command line.

Task definitions that contain errors (such as a misspelled option) are considered "invalid" tasks and will be reported in the task log along with an explanation of the error in the line. Examples of valid command lines are as follows:

```
# Commented line
/opt/framemaker/bin/assemble_frame --input /shared/scene23.def --time-index 0
cmd=/opt/framemaker/bin/assemble_frame --input /shared/scene23.def --time-index 0

name=Scene23Time0 /opt/framemaker/bin/assemble_frame --input /shared/scene23.def --
time-index 0
name=Scene23Time0 maxtime=30 cmd=/opt/framemaker/bin/assemble_frame --input
/shared/scene23.def --time-index 0
```

The following describes the various task options.

- **Application Command** – The coordinator considers everything immediately after the equal sign (= in "cmd=") as the task's "application" command line, which a worker will execute. There must be at least one non-whitespace character immediately after the "=" or the coordinator declares the task definition invalid. The application command line permits standard I/O redirection and environment variable substitution.

```
cmd=<xxx -y zzz>
```



Do not place any task options after the command line or the coordinator will not parse them; assumes they are part of the command line.

- **Labels** – Specifies the labels assigned to a task.

This is optional and there is no default value.

If given, a label must be composed of letters, digits, underscore, hyphen, and/or period. Use a comma to separate multiple labels.

If the option's value violates the conditions above, the coordinator will declare the task definition invalid and will not send the task to a worker.

When the coordinator logs the task in the Completed Tasks Log file, it outputs this option's value "as is", meaning without alternation and with no substitution of commas with spaces.

```
labels=<list>
```

- **Maximum Time** – Maximum time (in seconds) a task may execute after which the worker will terminate it. This is optional; the default value is 3,600 seconds (1 hour).

If given, the value must be less than the `maxtime-limit <period>` value. See [Nitro Configuration File on page 79](#) for more information on the `maxtime-limit` configuration option.

 If the option's value is non-numeric, non-decimal, or outside the allowed range, the coordinator will declare the task definition invalid and will not send the task to a worker.

```
maxtime=<nn>
```

- **Name** – Unique name assigned to your task definition. Task names do not have to be unique, but creating a unique task name will help to identify tasks.

```
name=<task name>
```

- **Task Cores** – Number of OS cores that the task requires. Nitro will allocate the number of cores requested and set the affinity of the task to the available cores.

```
cores=<count>
```

 The command line options "`--thread-count`" or "`--thread-ratio`" affect the number of available cores. If you use either of these options and they specify more cores than the node has available, Nitro will not pin the task to a specific core. If a task is specified to require more cores than the node that receives the task assignment, the task will not run.

 Users should submit their jobs so that tasks can run on any of the nodes that are allocated to the job. If, for example, you have some tasks that require 20 processors, but there are some 16 core nodes in the cluster, the job should be submitted so that it only allows 20 proc nodes to be allocated to the job. If a Nitro worker is assigned a job with requirements that it cannot fulfill (either too many cores, or too much memory) the task will be counted as failed, and Nitro will show the status of "InsufRes" for that task in the task log file.

- **Task Environment Variables** – Specifies a list of user supplied environment variables that will be set in the context of the task. The list of environment variables can be one or more name/value pairs separated by commas. Environment variable name value pairs cannot contain spaces.

```
env=<name=value>[, <name=value>, ...]
```

- **Task Memory** – Maximum amount of memory that the task requires. Nitro determines the amount of physical memory available on the system and uses this number as the limit that can be allocated by concurrent tasks. If no units are specified, GB is assumed. Available unit specifications include "GB" (10⁹ bytes), "GiB" (2³⁰ bytes), "MB" (10⁶ bytes), and "MiB" (2²⁰ bytes). Nitro uses MB units in debug logs.

```
memory=<amount>
```

i Users should submit their jobs so that tasks can run on any of the nodes that are allocated to the job. If, for example, you have some tasks that require 32 GB, but there are some 16 GB nodes in the cluster, the job should be submitted so that it only allows 32 GB nodes to be allocated to the job. If a Nitro worker is assigned a job with requirements that it cannot fulfill (either too many cores, or too much memory) the task will be counted as failed, and Nitro will show the status of "InsufRes" for that task in the task log file.

- **Task Shell** – Specifies the task shell, if any, to use. Tasks are normally executed by running `"/bin/bash -c <task command line>".`

```
shell=[default | none | <shell path>]
```

- The default shell provides translation of environment variables into command line options and other command line processing benefits.
- In high-throughput environments performance gains can be realized by using a lighter weight shell such as the Bourne shell or Korn shell.
- If no command line processing is needed, the task can be run without a shell.
- Executing a task directly instead of using the shell can speed task invocation by more than 50% over the default shell.

When specifying a shell other than the default shell, the fully qualified path should be used. For example, if you want to use the Bourne shell you should specify the shell as `"/bin/sh"` as opposed to just `"sh"`.

nitrostat

`nitrostat` is a utility found in the `/opt/nitro/bin` directory that will display the status of a Nitro job or of individual tasks. `nitrostat` lets you quickly find specific tasks or list all failed, invalid, or timed out tasks. `nitrostat` also offers a "wait" mode that will monitor the task log for tasks matching the specified criteria until the job completes.

Running nitrostat

To run nitrostat, you'll need to know the job ID of the job you want to monitor. For example, if you have a job with ID "3145", you can monitor the job progress with the following command:

```
/opt/nitro/bin/nitrostat 3145 -w
```

nitrostat assumes that the job information can be found in `$HOME/nitro/<job id>`.

If you have specified a different location for the job directory using the Nitro "`--job-dir`" command line option, then you'll need to specify the same location using the nitrostat "`--job-dir`" command line option.

For example, if your job directory is in `$HOME/projects/survey03` then use the following command:

```
/opt/nitro/bin/nitrostat 3145 --job-dir $HOME/projects/survey03 -w
```

nitrostat will show the following information when job status is requested:

```
Nitro Job Progress Report

Start Time   : 2015-06-17 09:10:11-0600
Current Time : 2015-06-17 09:10:42-0600
Elapsed Time : 31 seconds (00:00:31)

Job Id       : 23576
Coordinator  : node01
Task Log     : /home/jdoe/projects/survey03/23576/nitro_23576.tasklog.txt
Task File    : /home/jdoe/projects/survey03/survey03.tasks
  File Size  : 123366
  Est Tasks  : 3016
  Processed  : 75%

Tasks
-----
Pending      : 500
Running      : 500
Completed    : 1250
  Success    : 1250
  Failure    : 0
  InsufRes   : 0
  Timeout    : 0
  Invalid    : 0
  Tasks/sec  : 40.3
Total Tasks  : 2250

Workers
-----
Host  Port  Pid  Thrds Status  Assigned Running Completed  Success  Failure
InsufRes Timeout Tasks/sec AsgmtDur
node02 47000 6851 12  running 1250    250    1000    1000    0
0      0      36.0  8.0
node03 47000 14988 4  running 500     250    250     250     0
0      0      9.3   27.0
```

Searching for Task Records

You can use nitrostat to search the task log by task name, task ID, or label using regular expressions. You can also combine criteria to further refine your search.

For example, if you want to search for tasks containing the task name "Survey03" and the label "NYC" you can specify the command line as follows:

```
/opt/nitro/bin/nitrostat Job01 --name Survey03 --label NYC
```

The following identifies the nitrostat command line options.

- `--all, -a` – Shows all tasks.
- `--completed, -c` – Shows completed tasks.
- `--failed, -f` – Shows failed tasks.
- `--invalid, -i` – Shows invalid tasks.
- `--timedout, -o` – Shows tasks that timed out (exceeded maxtime).
- `--wait, -w` – Continues updating results until entire job is completed.
- `--name, -n <task name>` – Shows task(s) with the specified task name.
- `--task, -t <task id>` – Shows the task with the specified task ID.
- `label, -l <label list>` – Shows all tasks that contain the specified label. *<label list>* is a comma-separated list of labels that may *not* contain spaces.
- `--working-dir, -d <directory>` — Uses the specified working directory to locate the job and task log files. The default working directory is `$HOME/nitro`.
- `--regex` – If set, uses regular expression as the matching mode for *<task name>*, *<task id>*, and *<label list>*. The default is literal (exact string).

Job Recovery

Jobs run under a scheduler can, depending on job priority and settings, be preempted by a higher priority job, or even canceled by the user or administrator, or may fail due to hardware failure. Depending on the scheduler's configuration, a preempted job may be restarted later by the scheduler using the same job ID as the original job.

The job ID is the key to recovering jobs since Nitro uses the job ID as part of the path to the files associated with that job. Nitro tracks its progress by storing a checkpoint file that indicates which tasks have been completed and which have not. When Nitro is restarted, it looks for a checkpoint file and will continue from where it left off if one is found. If a job was canceled or preempted without a restart policy, then you will need to restart the job manually. Again, the key to restarting the job is to use the job ID of the original job.

The job ID is usually the ID that was returned when the job was submitted. There can be some differences between the scheduler's job ID and the resource manager's job ID depending on scheduler and resource manager settings. When you submitted your Nitro job, you may have set a Nitro job directory. If you didn't, it defaults to `$HOME/nitro/<jobid>`. This directory will contain the job log and task log files, along with checkpoint and Nitro log files. You can therefore use the directory name that Nitro created as the job directory with which to resubmit the job by passing the `--job-dir` option with the directory name through the `NITRO_OPTIONS` environment variable.

To restart the job you must set the `NITROJOBID` environment variable to the original job ID. Setting this environment variable will override the job ID provided by the resource manager and Nitro will resume from the line number of the task file described in the checkpoint file.

The checkpoint file is updated periodically when assignments are completed by workers and are returned to the coordinator. If a job is canceled, the workers will do their best to respond to the

coordinator with the tasks that have been completed so far, but depending on how quickly the resource manager forces the applications to close, the checkpoint file may or may not be fully updated. Therefore, it is possible that restarting a job will result in a particular task or set of tasks being run a second time. Users should take this into account and program their tasks so that if running the task a second time would cause a problem, transactions are recorded by the task that would prevent the second run.

If a job is canceled for reasons of task failure (for example, because of a typo in the task command line), you may want to submit the job as a new job instead of trying to resume the job with failed tasks.

i Failed and invalid tasks are marked as complete in the checkpoint file; they won't be re-run if the job is just restarted.

Coordinator Resiliency

Nitro has the ability to detect workers that have become unresponsive due to hardware, network, or software failure. During normal operation, workers periodically send an update to the coordinator. If the coordinator doesn't receive a status update after 45 seconds, the worker is deemed to be unresponsive, and any outstanding assignments will be revoked and reassigned to responsive workers. However, if the worker reports back to the coordinator before the assignment has been assigned to another worker, the assignment will be recovered and completed by that originally-assigned worker.

Glossary

Chapter 6 Troubleshooting

This chapter provides troubleshooting information.

In this topic:

- [Sources of Troubleshooting Information on page 93](#)
- [Troubleshooting Task Errors on page 93](#)

Related Topics

- [Track Job Progress on page 70](#)
- [nitrostat on page 89](#)

Sources of Troubleshooting Information

These are common sources of reference for troubleshooting:

- **Job Output Files** - Any errors that Nitro reports should be reported on stderr and will be captured to your job's output files (if running Nitro through a job scheduler).
- **Job and Task Log Files in the Job Directory** -
 - Nitro writes a job log indicating the startup parameters, input files, configuration, and the main worker events and statistics. You can review the job log to determine if any tasks failed, timed out, or were invalid (an error parsing the task line).
 - The task log contains a listing of all task results from the job and includes stdout and/or stderr output.
- **Nitro debug logs** - In the job directory you will also find a "logs" directory with worker and coordinator logs. The logs are named according to the role (worker or coordinator), host, job, and process ID so that logs being written to the same directory will not overwrite logs from another Nitro job or worker with the same process ID.

The default log level that Nitro logs at is level 3 (information), but this can be raised as high as 7 (debug) to gather much more information about the messages that Nitro is sending between the workers and coordinator, and the actions Nitro is taking (very large log file).

You can use the "--loglevel" command line parameter to set the log level independently on the workers and coordinator.

Related Topics

- [Troubleshooting on page 93](#)

Troubleshooting Task Errors

The Nitro job log, and the stdout output from the Nitro coordinator, lists the number of tasks completed, tasks successfully completed (exit code of 0), failed tasks (exit code other than 0), tasks that timed out (exceeded the "maxtime" task option), invalid tasks (tasks that the coordinator could not parse without errors), and tasks with insufficient resources.

i If you encounter a problem you are unable to solve, forward the log files according to your company's escalation process.

In this topic:

- [Task Command Line Errors on page 94](#)
- [Failed Tasks on page 94](#)
- [Invalid Tasks on page 95](#)
- [Insufficient Resources Tasks on page 95](#)

Task Command Line Errors

If the command line that you use to specify the task's command line contains an error (for example, the path is incorrect or you are attempting to run a script with noexecute permissions set), then the shell will output an error message to stderr that will be captured and stored in the task log file.

For example, if the task's command line references a binary that doesn't exist, you would see the following error in the task log file.

Job ID	Task ID	Line #	Task Name	Status	Ret	Hostname	Start time	Duration	UserCPU
SysCPU	VirtMem	PhysMem	Labels	Output					
EX01	3	3	S07T2303	Failure	127	node02	07:58:07.868	0.005	0.000
0.000	0	0		bash: /opt/framemaker/bin/framegen: No such file or directory					
EX01	4	4	S07T2304	Failure	127	node02	07:58:07.872	0.007	0.000
0.000	0	0		bash: /opt/framemaker/bin/framegen: No such file or directory					
EX01	5	5	S07T2305	Failure	127	node02	07:58:07.878	0.003	0.000
0.000	0	0		bash: /opt/framemaker/bin/framegen: No such file or directory					

Failed Tasks

Failed tasks were tasks that the worker executed, but have failed because the command line was not valid, or the task ran and returned an exit code other than 0. To diagnose the error, examine the task log file located in the job directory. See [Track Job Progress on page 70](#).

You can also use nitrostat to list failed tasks. See [nitrostat on page 89](#). To list failed tasks using nitrostat use the following command line.

```
/opt/nitro/bin/nitrostat <job id> -f
```

The following is an example of information provided by nitrostat showing failed tasks.

Job ID	Task ID	Line #	Task Name	Status	Ret	Hostname	Start time	Duration	UserCPU
SysCPU	VirtMem	PhysMem	Labels	Output					
EX01	3	3	S07T2303	Failure	127	node02	07:58:07.868	0.005	0.000
0.000	0	0		bash: /opt/framemaker/bin/framegen: No such file or directory					
EX01	4	4	S07T2304	Failure	127	node02	07:58:07.872	0.007	0.000
0.000	0	0		bash: /opt/framemaker/bin/framegen: No such file or directory					
EX01	5	5	S07T2305	Failure	127	node02	07:58:07.878	0.003	0.000
0.000	0	0		bash: /opt/framemaker/bin/framegen: No such file or directory					

The line number of the failed task in the task file is listed so you can easily identify which lines in the task file generated errors. If you need to verify the path to the task file used by Nitro you will find it both in the job log file and in the stdout output from the coordinator, which may also be recorded in the output files provided by your scheduler.

```
Nitro Environment
-----
Job Id       : EX01
Job path    : /home/jdoe/jobs/EX01
Job log     : /home/jdoe/jobs/EX01/nitro_EX01.joblog.txt
Task log    : /home/jdoe/jobs/EX01/nitro_EX01.tasklog.txt
Task file   : /home/jdoe/jobs/example1.tasks
Worker hosts : node02
```

If Nitro cannot access the task file, you will receive an error from the coordinator on stderr indicating that the task file was not found or is not accessible.

Invalid Tasks

Invalid tasks are lines in the task file that Nitro could not parse without errors. Parsing errors usually include misspelling a task option name. If your task line doesn't contain any task options, you should prepend the "cmd=" option to your command line. The "cmd" option indicates that Nitro should stop parsing the task line and accept the rest of the line as the command line to be executed. The following example shows an invalid task.

Job ID	Task ID	Line #	Task Name	Status	Ret	Output
EX01	14	14		Invalid	...	Unrecognized option name: walltime

In this case an invalid option "walltime" was used in a task definition instead of "maxtime".

```
# invalid line:
name=S07T2314 walltime=30 cmd=/opt/framemaker/bin/framegen -i /shared/scene07.def -
tindex 2314

# valid line:
name=S07T2314 maxtime=30 cmd=/opt/framemaker/bin/framegen -i /shared/scene07.def -
tindex 2314
```

The task file may also be rejected if you have any binary data in the file. The task file should only include ASCII text and each task must be on a separate line. The task file allows comment and empty lines. A hash symbol (#) in the first column of a line identifies a comment line.

Insufficient Resources Tasks

If workers are not able to fulfill resource requirements for tasks with cores or memory specifications, an insufficient resources error is logged. The following example shows a task with insufficient resources.

JobID	TaskID	Line	Name	Status	ExitCode	Hostname	StartTime
425	1	13		Success	0	node02	2016-02-16_17:58:07.052-
0700	0.030	0.010	0.000	130211840	1077248	1	1000000 3.142116000
425	2	14		Success	0	node02	2016-02-16_17:58:07.052-
0700	0.030	0.010	0.000	130211840	1081344	2	1000000 3.141296000
425	3	15		Success	0	node02	2016-02-16_17:58:07.083-
0700	0.021	0.010	0.000	275505152	1679360	3	1000000 3.139544000
425	4	16		InsufRes	-1	node02	2016-02-16_17:58:07.083-
0700	0.000	0.000	0.000		0		Error: worker :
to 2 threads, task is requesting 3 threads							

Related Topics

- [Track Job Progress on page 70](#)
- [nitrostat on page 89](#)
- [Troubleshooting on page 93](#)