# Core Extended Monitoring

Zenoss, Inc.

www.zenoss.com

# Zenoss Core Extended Monitoring



Part Number: 03-062013-4.2-v05

1. ZenPacks ..... 1
1.1. About ZenPacks ..... 1
1.1.1. Working with ZenPacks ..... 1
1.2. Viewing Loaded ZenPacks ..... 1
1.3. Installing ZenPacks ..... 2
1.3.1. Installing from the Command Line ..... 2
1.3.2. Installing from the User Interface ..... 3
1.4. Creating ZenPacks ..... 3
1.4.1. Why Create a ZenPack? ..... 3
1.4.2. Create a ZenPack ..... 4
1.4.3. Add a Database Object to a ZenPack ..... 4
1.4.4. View Database Objects in a ZenPack ..... 5
1.4.5. Remove a Database Object from a ZenPack ..... 5
1.4.6. Adding Other Items to ZenPacks ..... 5
1.5. Packaging ZenPacks ..... 5
1.6. Distributing ZenPacks ..... 6
1.7. Removing ZenPacks ..... 6
1.8. Where to Find More Information ..... 7
I. Zenoss Core ZenPacks ..... 8
2. Amazon Web Services ..... 9
2.1. About ..... 9
2.2. Prerequisites ..... 9
2.3. Features ..... 9
2.3.1. Discovery of EC2 Entities ..... 9
2.3.2. Monitoring ..... 10
2.3.3. Guest Device Discovery ..... 11
2.3.4. Service Impact ..... 11
2.4. Setup ..... 11
2.5. Working with the EC2Manager Account ..... 13
2.5.1. CloudWatch Data ..... 13
2.5.2. Templates and Collection ..... 14
2.5.2.1. Example: Initiating Load-Based Elasticity for an EC2 Setup ..... 14
3. Apache Web Server ..... 15
3.1. About ..... 15
3.2. Prerequisites ..... 15
3.3. Enable Monitoring ..... 15
3.3.1. Display the Status Page in Apache Version 1.3 or Higher ..... 15
3.3.2. Display the Status Page in Apache Version 2.x ..... 17
3.3.3. Verifying Your Apache Configuration ..... 18
3.3.4. Configure Zenoss Core to Monitor the Web Server ..... 18
3.4. Daemons ..... 19
4. Dell Hardware ..... 20
4.1. About ..... 20
4.2. Prerequisites ..... 20
4.3. Enable Enhanced Modeling ..... 20
4.4. Daemons ..... 21
5. Device Search ..... 22
5.1. About ..... 22
5.2. Prerequisites ..... 22
6. Domain Name System (DNS) ..... 23
6.1. About ..... 23
6.2. DigMonitor ..... 23
6.2.1. Prerequisites ..... 23
6.2.2. Enable Monitoring ..... 23

6.3. DNSMonitor ..... 23
6.3.1. Prerequisites ..... 24
6.3.2. Enable Monitoring ..... 24
6.4. Daemons ..... 24
7. File Transfer Protocol (FTP) ..... 25
7.1. About ..... 25
7.2. Prerequisites ..... 25
7.3. Enable Monitoring ..... 25
7.4. Enable Secure Site Monitoring ..... 25
7.5. Tuning for Site Responsiveness ..... 26
7.6. Daemons ..... 26
8. HP PC Hardware ..... 27
8.1. About ..... 27
8.2. Prerequisites ..... 27
8.3. Enable Enhanced Modeling ..... 27
8.4. Daemons ..... 28
9. Internet Relay Chat (IRC) ..... 29
9.1. About ..... 29
9.2. Prerequisites ..... 29
9.3. Enable Monitoring ..... 29
9.4. Daemons ..... 29
10. Jabber Instant Messaging ..... 30
10.1. About ..... 30
10.2. Prerequisites ..... 30
10.3. Enable Monitoring ..... 30
10.4. Daemons ..... 31
11. Java 2 Platform Standard Edition (J2E) ..... 32
11.1. About ..... 32
11.1.1. JMX Background ..... 32
11.1.2. ZenJMX Capabilities ..... 32
11.1.3. Allowable Parameter Types ..... 33
11.1.4. Single Value Attribute Calls ..... 33
11.1.5. Complex-Value Attribute Calls ..... 34
11.1.6. Example Method Calls ..... 34
11.1.6.1. No parameters, single return value ..... 35
11.1.6.2. No parameters, multiple values returned in List format ..... 35
11.1.6.3. No parameters, multiple values returned in Map format ..... 35
11.1.6.4. Single parameter in polymorphic operation ..... 36
11.1.6.5. Multiple parameters in polymorphic operations ..... 36
11.1.7. Special Service URLs ..... 37
11.2. Prerequisites ..... 37
11.2.1. Oracle Java Runtime Environment (JRE) ..... 37
11.3. Example to Monitor a JMX Value ..... 38
11.3.1. Enabling Remote JMX Access ..... 38
11.3.2. Configure Zenoss Core with a Custom Data Source ..... 38
11.4. Monitor Values in TabularData and CompositeData Objects ..... 40
11.5. Using **JConsole** to Query a JMX Agent ..... 40
11.6. ZenJMX Options ..... 44
11.7. Memory Allocation ..... 44
11.8. ZenJMX Logging ..... 44
11.9. Daemons ..... 45
12. Lightweight Directory Access Protocol (LDAP) Response Time ..... 46
12.1. About ..... 46
12.2. Prerequisites ..... 46

12.3. Enable Monitoring ..... 46
12.3.1. For a Device ..... 46
12.4. Daemons ..... 47
13. MySQL Database ..... 48
13.1. About ..... 48
13.2. Prerequisites ..... 48
13.3. Enable Monitoring ..... 48
13.3.1. Authorize MySQL Performance Data Access ..... 48
13.3.2. Set up Zenoss Core ..... 48
13.4. Daemons ..... 49
14. Network News Transport Protocol (NNTP) ..... 50
14.1. About ..... 50
14.2. Prerequisites ..... 50
14.3. Enable Monitoring ..... 50
14.4. Daemons ..... 50
15. Network Time Protocol (NTP) ..... 51
15.1. About ..... 51
15.2. Prerequisites ..... 51
15.3. Enable Monitoring ..... 51
15.4. Daemons ..... 51
16. ONC-Style Remote Procedure Call (RPC) ..... 52
16.1. About ..... 52
16.2. Prerequisites ..... 52
16.3. Enable Monitoring ..... 52
16.4. Daemons ..... 52
17. SSH Monitoring Example ..... 53
17.1. About ..... 53
17.2. Prerequisites ..... 53
17.3. Set Linux Server Monitoring Credentials ..... 53
17.4. Add a Linux Server ..... 53
17.5. Daemons ..... 54
18. VMware esxtop ..... 55
18.1. About ..... 55
18.2. Prerequisites ..... 55
18.2.1. Installing Prerequisite Libraries ..... 55
18.3. Enabling the ZenPack ..... 56
18.4. Daemons ..... 57
19. Web Page Response Time (HTTP) ..... 58
19.1. About ..... 58
19.2. Prerequisites ..... 58
19.3. Enable Monitoring ..... 58
19.4. Check for a Specific URL or Specify Security Settings ..... 58
19.5. Check for Specific Content on the Web Page ..... 59
19.6. Tuning for Site Responsiveness ..... 60
19.7. Daemons ..... 60
20. Xen Virtual Hosts ..... 61
20.1. About ..... 61
20.2. Prerequisites ..... 61
20.3. Model Hosts and Guest ..... 61
20.4. Daemons ..... 61

# Chapter 1. ZenPacks

## 1.1. About ZenPacks

ZenPacks provide a plug-in architecture that extends and modifies the system to add new functionality. This can be as simple as adding new device classes or monitoring templates, or as complex as extending the data model and providing new collection daemons.

You can use ZenPacks to add:

- Monitoring templates
- Data sources
- Graphs
- Event classes
- Event and user commands
- Reports
- Model extensions
- Product definitions

Simple ZenPacks can be created completely within the user interface. More complex ZenPacks require development of scripts or daemons, using Python or another programming language.

ZenPacks can be distributed for installation on other Zenoss Core systems.

### 1.1.1. Working with ZenPacks

A range of provided ZenPacks add and extend system functionality. The following sections provide information and procedures to help you:

- View, install, and create ZenPacks
- Package and distribute ZenPacks
- Remove ZenPacks

## 1.2. Viewing Loaded ZenPacks

To see which ZenPacks are loaded on your system:

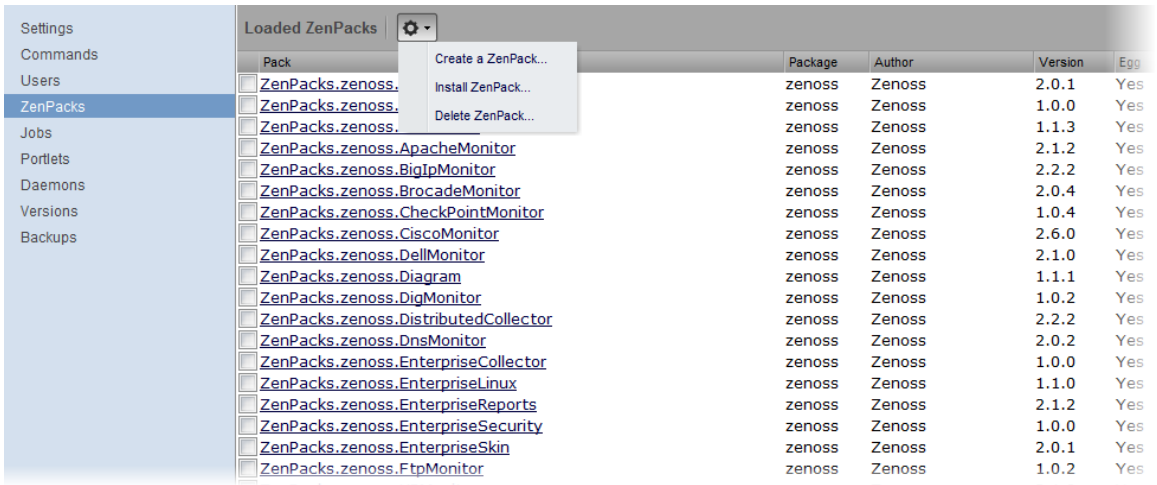
1. From the navigation bar, select Advanced.

   The Settings page appears.

2. Select ZenPacks in the left panel.

   The list of loaded ZenPacks appears.

**Figure 1.1. Loaded ZenPacks**




From the action menu on this page, you can create, install, and delete ZenPacks.

### Note

Alternatively, you can view loaded ZenPacks from the command line:

```
zenpack --list
```

# 1.3. Installing ZenPacks

ZenPacks are distributed as `.egg` files. You can install ZenPacks from the command line on the Zenoss Core server, or from the user interface.

### Note

You can find free, open source ZenPack eggs at the Zenoss Community ZenPacks page:

http://community.zenoss.org/community/zenpacks

## 1.3.1. Installing from the Command Line

Use these commands to install a ZenPack file, and then restart the system:

```
zenpack --install <filename>
zenoss restart
```

If you have the source code for the ZenPack you can install directly from that rather than from an `.egg` file. The command is the same; however, you must specify the directory containing the source code. This copies the source code to `$ZENHOME/ZenPacks`:

```
zenpack --install <directoryname>
zenoss restart
```

If you are developing a ZenPack, you should maintain your source code outside of `$ZENHOME/ZenPacks` for two reasons:

- if you issue a **zenpack --remove** command it will delete your code from that location and you will lose your files unless you have them backed up elsewhere.

- if you are maintaining your source code in a version control system it is frequently more convenient to have the files reside elsewhere on the file system.

Using the **--link** option, you can install the ZenPack but have the system use your code from its current location. Instead of installing your code in `$ZENHOME/ZenPacks`, the system will create a link in that location that points to your source code directory.

```
zenpack --link --install <directoryname>
zenoss restart
```

This option is useful for testing your ZenPacks. When you develop a ZenPack outside of `$ZENHOME/ZenPacks`, you may want your changes to be caught by the version control system and installed in place, so that future edits affect the installed ZenPack.

### 1.3.2. Installing from the User Interface

To upload and install a ZenPack `.egg` file from the user interface:

1. From the navigation bar, select Advanced > Settings.
2. In the left panel, select ZenPacks.
3. From  (Action menu), select Install ZenPack.

   The Install ZenPack dialog appears.
4. Browse to and select the `.egg` file you want to install, and then click **OK**.

   The file is uploaded to the Zenoss Core server and installed.

   #### Note

   After installing the ZenPack, you should restart the system.

## 1.4. Creating ZenPacks

Read the following information and procedures to learn more about why you might want to create a ZenPack, and how to:

- Create a ZenPack
- Add a database object to a ZenPack
- View database objects in a ZenPack
- Remove a database object from a ZenPack
- Add other items to a ZenPack

### 1.4.1. Why Create a ZenPack?

Suppose you have developed a monitoring template for a new piece of hardware. You have created data sources for the OID's you think are worth monitoring, thresholds to make sure some of these values stay within reasonable limits, and several graph definitions to show this data graphically. Perhaps you also have created a new device class for this hardware. You can create a ZenPack to easily distribute your template and device class to other administrators. This ZenPack can be entirely created from within the user interface.

As another example, suppose you want to monitor a new piece of software running on one of your servers. You would like to monitor several performance metrics of this software, but they are available only via a programmatic API

provided with the software. You could develop a new collector daemon to gather data via this API and provide it back to the system. You might also create a new type of data source to provide configuration data for the new collector. Obviously this effort would require development skills and intimate knowledge of the system not necessary for the previous example, but this functionality can be distributed as a ZenPack.

## 1.4.2. Create a ZenPack

Use the following instructions and guidelines to create a ZenPack.

### Note

You must be logged in as an administrator to create a ZenPack.

1. From the navigation bar, select Advanced > Settings.
2. In the left panel, select ZenPacks.
3. From  (Action menu), select Create a ZenPack.

   The Create a ZenPack dialog appears.
4. Enter the name of the ZenPack, which must be in the format:

   ZenPacks.*Organization*.*Identifier*

   where *Organization* is a name that identifies you or your organization and *Identifier* is a string that represents the intent of your ZenPack.

   ### Note

   Do not use underscores in ZenPack names. ZenPacks whose names include an underscore break on export.
5. Click **OK**.

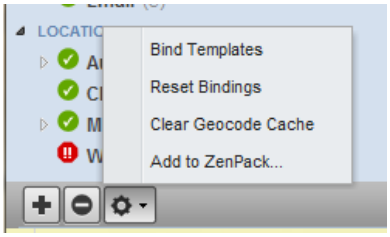
   The system creates the ZenPack object in the database and a new directory in the file system `$ZENHOME/ZenPacks/`*YourZenPackID*.

## 1.4.3. Add a Database Object to a ZenPack

To add a database object (such as a device, service, or event class, event mapping, user or event command, device organizer, or monitoring template) to a ZenPack:

1. Navigate to the object in the interface.
2. From the Action menu, select Add to ZenPack.

### Figure 1.2. Add to ZenPack



The Add to ZenPack dialog appears.

3. Select a ZenPack from the list of installed ZenPacks, and then click **Submit**.

### 1.4.4. View Database Objects in a ZenPack

To view the objects that are part of a ZenPack:

1. From the navigation bar, select Advanced > Settings.
2. In the left panel, select ZenPacks.
3. Click the name of a ZenPack in the list.

   The ZenPack Provides area of the page lists objects that are part of the ZenPack.

### 1.4.5. Remove a Database Object from a ZenPack

To remove a database object from a ZenPack:

1. From the navigation bar, select Advanced > Settings.
2. In the left panel, select ZenPacks.
3. Click the name of a ZenPack in the list.
4. Select an object in the ZenPack Provides area of the page.
5. From the Action menu, select Delete from ZenPack.

### 1.4.6. Adding Other Items to ZenPacks

ZenPacks can contain items that are not database items, such as:

- Daemons
- Data source types
- Skins

You can add these to a ZenPack by placing them in the appropriate subdirectory in the ZenPack's directory.

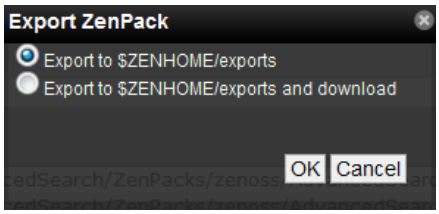
## 1.5. Packaging ZenPacks

Follow these steps to create an installable ZenPack .egg file:

1. From the navigation bar, select Advanced > Settings.
2. In the left panel, select ZenPacks.
3. Click the name of a ZenPack in the list.
4. From  (Action menu) located at the bottom left of the page, select Export ZenPack.

   The Export ZenPack dialog appears.

**Figure 1.3. Export ZenPack**




5. Select one of the export options:

   - **Export to $ZENHOME/exports** - Exports the ZenPack to a file named *ZenPackID*`.egg` in the `$ZENHOME/exports` directory on the Zenoss Core server.
   - **Export to $ZENHOME/exports** - Additionally downloads the exported file to your browser. Other administrators can then install this exported `.egg` file.

6. Click **OK**.

# 1.6. Distributing ZenPacks

Zenoss encourages you to distribute ZenPacks to the Zenoss open source community. To make a ZenPack publicly available, follow the ZenPack Development Process instructions posted to this location:

http://community.zenoss.org/docs/DOC-8495

Installable `.egg` files are supplied at the Zenoss Community ZenPacks page:

http://community.zenoss.org/community/zenpacks

ZenPack code is posted to:

https://github.com/zenoss/

# 1.7. Removing ZenPacks

Deleting a ZenPack can have unexpected consequences. Before you take this action, consider that:

- Deleting a ZenPack will delete all objects provided by the ZenPack, as well as all objects that depend on code provided by the ZenPack. Zenoss recommends that you view a ZenPack's detail page to see its dependencies, files, and classes before removing it.
- Removing a ZenPack that installs a device class removes the device class, any contained device classes, and all devices in that class.

Before removing a ZenPack, you should:

- Delete any data source of a type provided by the ZenPack
- Back up your system data. (See the section titled "Backup and Restore" in *Zenoss Core Administration* for information on backing up system data.)

To delete a ZenPack:

1. Select the ZenPack from the Loaded ZenPacks list.

2. Select Delete ZenPack from the Action menu.
3. Click **OK** to confirm removal.

### Note

If the ZenPack includes daemons, you must manually stop these after deleting the ZenPack.

## 1.8. Where to Find More Information

Further information about ZenPack development is available in the *Zenoss Developer's Guide*.

Discussions about ZenPack development and implementation take place on these forums:

- http://community.zenoss.org/community/developers/zenpack_development
- http://community.zenoss.org/community/forums
- https://github.com/zenoss

# Part I. Zenoss Core ZenPacks

# Chapter 2. Amazon Web Services

## 2.1. About

The Amazon Web Services™ ZenPack allows you to monitor Amazon Elastic Compute Cloud™ (Amazon EC2™) server instances. It collects information for these objects monitored through a combination of AWS EC2 and Cloud-Watch APIs:

When you install the ZenPack, the `/AWS/EC2` device class is added to your Zenoss Core instance. A single device in the EC2 class, `EC2Manager`, represents your EC2 account. All instances and instance types are contained in the EC2 account manager.

## 2.2. Prerequisites

You must have a valid Amazon Web Services account with the Elastic Compute Cloud service enabled.

Modeling and performance requests to Amazon are sent via XML over http or https. You must open port 80, port 443, or both on your Zenoss Core server so that requests can be sent to Amazon's infrastructure through the Internet.

The Zenoss Core server time must be correct; otherwise, no performance data will be returned.

**Table 2.1. Prerequisites**

| Prerequisite | Restriction |
|---|---|
| Product | Zenoss Core Version 4.2.x |
| Required ZenPacks | ZenPacks.zenoss.AWS<br><br>ZenPacks.zenoss.PythonCollector |

### Note

The ZenPacks.zenoss.AWS supercedes the older ZenPacks.zenoss.ZenAWS that was installed by default on versions of Zenoss prior to version 4.2.4. Please remove the ZenAWS ZenPack before installing ZenPacks.zenoss.AWS.

## 2.3. Features

The following features are available in this ZenPack:

- Discovery of EC2 entities
- Monitoring of CloudWatch metrics
- Optional auto-discovery and monitoring of instance guest operating systems
- Optional service impact with addition of Zenoss Service Dynamics product

### 2.3.1. Discovery of EC2 Entities

The following entities will be automatically discovered through an account name, access key and secret key you provide. The attributes, tags and collections will be updated on Zenoss' normal remodeling interval which defaults to every 12 hours.

- Regions

  Attributes: ID

  Collections: VPCs, Subnets, Zones, Instances, Volumes

- Zones

  Attributes: ID, Region, State

  Collections: Instances, Volumes, Subnets

- VPCs

  Attributes: ID, Region, CIDR Block

  Tags: Name, Collector

  Collections: Subnets, Instances

- Subnets

  Attributes: ID, Region, VPC, Zone, State, CIDR Block, Available IP Address Count, Zone Default, Auto-Public IP

  Tags: Name

  Collections: Instances

- Instances

  Attributes: ID, Region, VPC, Zone, Subnet, State, Instance Type, Image ID, Platform, Public DNS Name, Private IP Address, Launch Time, Guest Device

  Tags: Name

  Collections: Volumes

  Other: Guest Device (if monitored by Zenoss)

- Volumes

  Attributes: ID, Region, Zone, Instance, Type Created Time, Size, IOPS, Status, Attach Data Status, Attach Data Device

  Tags: Name

## 2.3.2. Monitoring

The following metrics will be collected every 5 minutes by default. Any other CloudWatch metrics can also be collected by adding them to the appropriate monitoring template. The *Average* statistic is collected, and the graphed value is per second for anything that resembles a rate. The *Amazon CloudWatch* datasource type also allows for the collection of any other CloudWatch metric.

- Regions

  Metrics: CPUUtilization, DiskReadOps, DiskWriteOps, DiskReadBytes, DiskWriteBytes, NetworkIn, NetworkOut

- Instances

Metrics: CPUUtilization, DiskReadOps, DiskWriteOps, DiskReadBytes, DiskWriteBytes, NetworkIn, NetworkOut, StatusCheckFailed_Instance, StatusCheckFailed_System

- Volumes

  Metrics: VolumeReadBytes, VolumeWriteBytes, VolumeReadOps, VolumeWriteOps, VolumeTotalReadTime, VolumeTotalWriteTime, VolumeIdleTime, VolumeQueueLength Provisioned IOPS Metrics: VolumeThroughputPercentage, VolumeReadWriteOps

### 2.3.3. Guest Device Discovery

You can optionally configure each monitored AWS account to attempt to discover and monitor the guest Linux or Windows operating systems running within each EC2 instance. This requires that your Zenoss system has the network and server access it needs to monitor the guest operating system. VPC and non-VPC modes are supported.

The guest operating system devices' life-cycle are managed along with the instance. For example, the guest operating system device is set to a decommissioned production state when the EC2 instance is stopped, and the guest operating system device is deleted when the EC2 instance is destroyed.

### 2.3.4. Service Impact

When combined with the Zenoss Service Dynamics product, this ZenPack adds built-in service impact capability for services running on AWS. The following service impact relationships are automatically added. These will be included in any services that contain one or more of the explicitly mentioned entities.

- Account access failure impacts all regions.
- Region failure affects all VPCs and zones in affected region.
- VPC failure affects all related subnets.
- Zone failure affects all related subnets, instances, and volumes.
- Subnet failure affects all instances on affected subnet.
- Volume failure affects any attached instance.
- Instance failure affects the guest operating system device.

## 2.4. Setup

To set up the EC2 account manager in Zenoss Core, follow these steps:

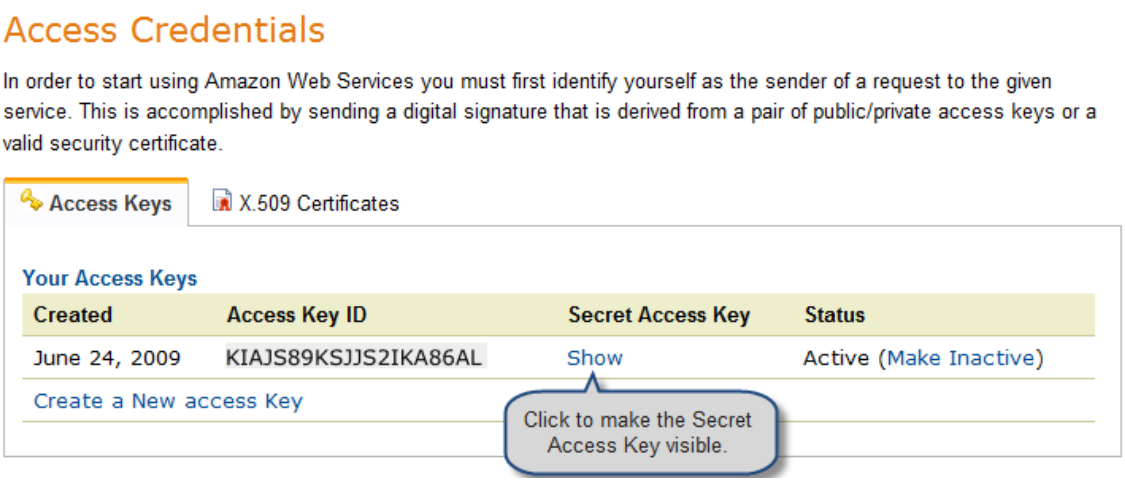
1. Retrieve your Amazon EC2 access credentials.

   a. Browse to http://aws.amazon.com.

   b. Select **Security Credentials** from the **Your Account** list of options.

      The Access Key ID and Secret Access Key values appear on the Access Keys tab.

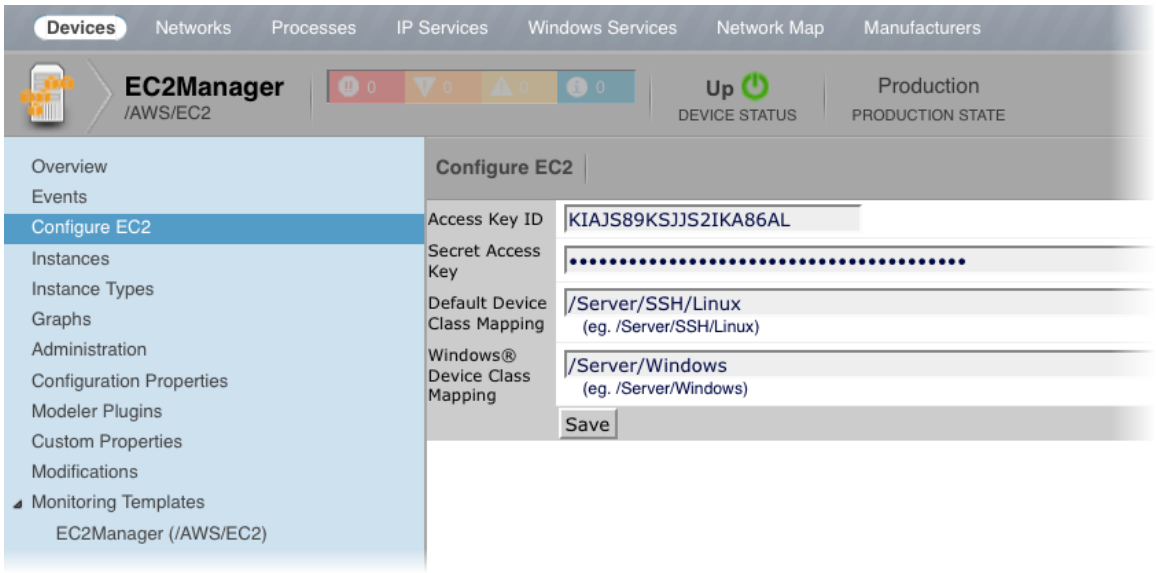
**Figure 2.1. Access Credentials**




2. In the Zenoss Core interface, select Infrastructure, and then select the EC2Manager device in the device list.
3. Select Configure EC2 in the left panel.

   The Configure EC2 page appears.

**Figure 2.2. Configure EC2**



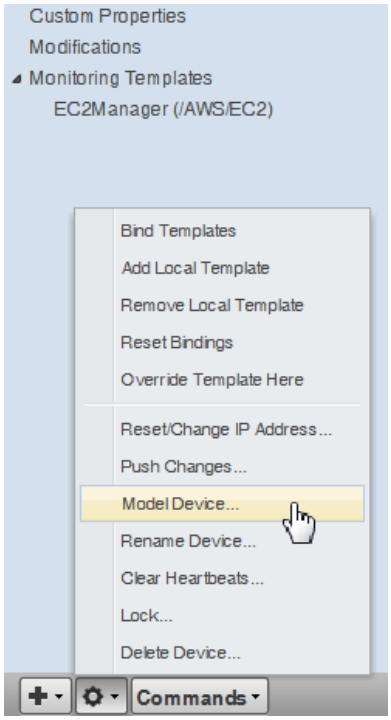

4. Enter access credentials.

   ## Note

   Entering a class for the Device Mapping field allows the system to monitor an EC2 instance as a normal device. If no class is specified, then the instances are monitoring within EC2Manager only.

5. Model the EC2Manager account to retrieve the Instance and InstanceType objects. From the Action menu, select Model Device.

**Figure 2.3. Model EC2Manager Account**




Alternatively you can use zenbatchload to add AWS accounts from the command line. To do this, you must create a file with contents similar to the following. Replace all values in angle brackets with your values minus the brackets. Multiple accounts can be added under the same */Device/AWS/EC2* section.

```
/Devices/AWS/EC2 loader='ec2account', loader_arg_keys=['accountname', 'accesskey', 'secretkey', 'collector']
<accountname> accountname='<accountname>', accesskey='<accesskey>', secretkey='<secretkey>', collector='<col
```

You can then load the account(s) with the following command:

```
$ zenbatchload <filename>
```

# 2.5. Working with the EC2Manager Account

Select Infrastructure, and then select the EC2Manager account in the device list. Select Instances in the left panel to see each instance that is active in your Amazon EC2 account. Click an instance to view detailed information

The Instance Type field is a link to a type object that references all instances of a particular type.

## 2.5.1. CloudWatch Data

Amazon provides monitoring information through its CloudWatch APIs. These APIs provide monitoring information for each of their primary objects (Account, Instance, and Instance Type).

Metrics provided by the API are:

- CPU utilization
- Network in/out
- Disk bytes read/write

- Disk operations read/write

The metrics for an instance apply directly for the instances; for example, if an instance shows 100% CPU utilization, then its CPU is at maximum. However, for an instance type, 100% CPU utilization means that all instances within that type are at 100% CPU utilization. The same is true for the account; metrics are summed for all instances.

Zenoss Core collects monitoring information for the Account, Instance, and Instance Type objects. Account information appears on the Perf tab. Instance and Instance Type information appears on their main screens.

## 2.5.2. Templates and Collection

Zenoss Core uses the standard monitoring template system to configure EC2 Manager accounts. Each template relies on a custom ZenCommand, `zencw2`, that polls the CloudWatch API and returns all available parameters. These parameters are used by their associated graphs. You can set thresholds against the parameters.

Templates for each primary object type are defined in the `/AWS/EC2` class.

### Table 2.2. Primary Object Type Templates

| Object Type | Template |
|---|---|
| Account | EC2Manager |
| Instance | EC2Instance |
| Instance Type | EC2InstanceType |

### 2.5.2.1. Example: Initiating Load-Based Elasticity for an EC2 Setup

Suppose you want to use Zenoss Core to initiate load-based elasticity for your EC2 setup. For example, each time the account CPU exceeds 80%, you want Zenoss Core to create a new instance. To set up this scenario, you would first set up and model your account. Then, you would follow these steps:

1. Select the EC2Manager device in the Devices section of the Infrastructure page, and then expand the Monitoring templates node at the left of screen and click the EC2Manager template.

2. Add a threshold against the `zencw2_CPUUtilization` CPU Utilization data point, and then set its event class to `/Perf/CPU`.

   Each time the CPU exceeds the threshold, Zenoss Core creates an event with the device name EC2Manager in the `/Perf/CPU` class.

3. Create an event command that matches this event, and launch the EC2 command to create the new instances.

   When the event is initiated, the new instances will be created.

   #### Note

   The `clear` command can be used to shut down unneeded instances.

# Chapter 3. Apache Web Server

## 3.1. About

The ApacheMonitor ZenPack provides a method for pulling performance metrics from the Apache Web server directly into Zenoss Core, without requiring the use of an agent. This is accomplished by using Apache's `mod_status` module that comes with Apache Version 1 and 2.

The following metrics are collected and graphed for the Apache HTTP server.

- Requests per Second
- Throughput (Bytes/sec and Bytes/request)
- CPU Utilization of the HTTP server and all worker processes or threads
- Slot Usage (Open, Waiting, Reading Request, Sending Reply, Keep-Alive DNS Lookup, and Logging)

## 3.2. Prerequisites

**Table 3.1. Apache Prerequisites**

| Prerequisite | Restriction |
|---|---|
| Product | Zenoss Core Version 2.2 or higher |
| Required ZenPacks | ZenPacks.zenoss.ApacheMonitor |

## 3.3. Enable Monitoring

Follow the steps in these sections to:

- Display the status page in Apache Version 1.3 or higher
- Display the status page in Apache Version 2.x
- Configure your configuration
- Configure the system to monitor the Web server

### 3.3.1. Display the Status Page in Apache Version 1.3 or Higher

1. On the Apache server, locate the `httpd.conf` file. Generally, this file is located at `/etc/httpd/httpd.conf` or `/etc/httpd/conf/httpd.conf`; however, other locations are possible depending on your operating system and setup.

   If you cannot locate the configuration file, use your system's search facilities to locate it. For Windows, use the Search button of the Windows Explorer tool. For Unix, try the following command:

   ```
   find / -name httpd.conf
   ```

2. Check to see that the following line is not commented out and is available in `httpd.conf` or `/etc/apache/modules.conf`:

   ```
   LoadModule status_module /usr/lib/apache/1.3/mod_status.so
   ```

### Note

You may have to search in alternate locations to find the `mod_status.so` file. Also, the syntax may differ depending on your configuration.

3. Turn the `ExtendedStatus` option on in the `httpd.conf file`. This option is typically commented out. You can enable it by uncommenting it or ensuring that it is defined.

```
#ExtendedStatus on
```

becomes:

```
ExtendedStatus on
```

4. Enable the `/server-status` location in the `httpd.conf` file. Typically, this option exists but is commented out.

```
#<Location /server-status>
#    SetHandler server-status
#    Order deny,allow
#    Deny from all
#    Allow from .example.com
#</Location>
```

becomes:

```
<Location /server-status>
SetHandler server-status
Order deny,allow
Deny from all
Allow from zenoss.example.com
</Location>
```

### Note

Your Zenoss Core server or servers must be able to connect to your Apache server. Ensure that it is listed here or is part of the network specified in this chunk of configuration.

To specify multiple servers, separate the entries with spaces. If you specify an IP address range rather than a destination, be sure to add a network mask to the end of the IP address range.

The following example allows a server called `externalzenoss.example.com`, as well as all servers that start with 192.168.10, in their addresses:

```
<Location /server-status>SetHandler server-status
Order deny,allow
Deny from all
Allow from externalzenoss.example.com 192.168.10.0/24
</Location>
```

5. Save the `httpd.conf` file with these changes and verify that the configuration file is correct. This can be accomplished with following command.

```
apachectl -t
```

Correct any issues before restarting Apache.

6. Restart the Web server (`httpd`). This can be accomplished with following command.

```
apachectl restart
```

## 3.3.2. Display the Status Page in Apache Version 2.x

1. On the Apache server, find the `httpd.conf` file. This is usually `/etc/apache2/apache2.conf` or `/etc/apache2/conf/httpd.conf`; however, other locations are possible depending on your operating system and setup.

   If you are unsure about where your configuration file is located, use your system;s search facilities to locate this file. Under Windows, use the Search button of the Windows Explorer tool. Under Unix, try the following command:

   ```
   find / -name httpd.conf
   ```

2. Verify that the `mod_status` module is loaded.

   ```
    apache% apachec2ctl -M 2<&1 | grep status
   status_module (shared)
   ```

   The previous output indicates that the module is loaded and no further configuration is necessary. If there is no output, then copy the `mods-available/status.load` to the `mods-enabled` directory, and then run:

   ```
    apache% /etc/init.d/apache2 force-reload
   ```

3. Turn the ExtendedStatus option on in the `httpd.conf file`. This option is typically commented out. You can enable it by uncommenting it or ensuring that it is defined.

   ```
   #ExtendedStatus on
   ```

   becomes:

   ```
   ExtendedStatus on
   ```

4. Enable the `/server-status` location in the `httpd.conf` file. This is another option that typically already exists but is commented out.

   ```
   #<Location /server-status>
   #    SetHandler server-status
   #    Order deny,allow
   #    Deny from all
   #    Allow from .example.com
   #</Location>
   ```

   becomes:

   ```
   <Location /server-status>
   SetHandler server-status
   Order deny,allow
   Deny from all
   Allow from zenoss.example.com
   </Location>
   ```

   ### Note

   Your Zenoss Core server or servers must be able to connect to your Apache server so you must ensure that it is either listed here or is a part of the network specified in this chunk of configuration.

   To specify multiple servers, separate the entries with spaces. If you would like to specify an IP address range rather than a destination, be sure to add a network mask to the end of the IP address range. The following example allows a server called `externalzenoss.example.com` as well as all servers that start with '192.168.10' in their addresses:

   ```
   <Location /server-status>SetHandler server-status
   Order deny,allowDeny from all
   Allow from externalzenoss.example.com 192.168.10.0/24
   ```

```
</Location>
```

5. Save the `httpd.conf` file with these changes and verify that the configuration file is correct. This can be accomplished with following command.

   ```
   apache2ctl -t
   ```

   Correct any issues before restarting Apache.

6. Restart the webserver (`httpd`). This can be accomplished with following command.

   ```
   apache2ctl restart
   ```

## 3.3.3. Verifying Your Apache Configuration

Once Apache has been configured, you should verify that it is working correctly. To verify your Apache server, point your Web browser to your Apache server at the appropriately modified URL:

`http://`*`your-apache-server`*`/server-status?auto`

This is an example of what you might see:

```
Total Accesses: 1
Total kBytes: 2
Uptime: 43
ReqPerSec: .0232558
BytesPerSec: 47.6279
BytesPerReq: 2048
BusyWorkers: 1
IdleWorkers: 5
Scoreboard: _W____.............................
```

If there is a configuration issue, you should see an error message telling you that the page is forbidden.

### Note

Your Zenoss Core server or servers must be able to connect to your Apache server by using HTTP to receive information. This means that the server must be permitted not only by the Apache configuration settings, but also by any firewalls or proxies between them and the Apache server, including any firewall on the Apache server. If there are any proxies, they must be configured to allow the Zenoss Core HTTP traffic through. Consult your network administrator and security officer to verify the firewall configuration and your site's policies.

Further note that the name or IP address that your server has behind a firewall may be different than the IP address (some form of Network Address Translation (NAT)) or name resolution (the way that the external server resolves names may be through an Internet-visible DNS system rather than an intranet-only DNS system).

## 3.3.4. Configure Zenoss Core to Monitor the Web Server

Once the Apache server is configured to allow Zenoss Core to access the extended status, you can add Apache monitoring to the device within Zenoss Core by binding the Apache template to the device.

1. Select Infrastructure from the navigation bar.

2. Click the device name in the device list.

   The device overview page appears.

3. In the left panel, expand Monitoring Templates, and then select Device.
4. Select Bind Templates from the Action menu.

   The Bind Templates dialog appears.
5. Add the Apache template to the list of templates, and then click **Save**.

   The Apache template is added. The system can now begin collecting the Apache server metrics from this device.

# 3.4. Daemons

### Table 3.2. Daemons

| Type | Name |
|---|---|
| Performance Collector | **zencommand** |

# Chapter 4. Dell Hardware

## 4.1. About

The DellMonitor ZenPack provides custom modeling of devices running the Dell OpenManage agents. It also contains hardware identification for Dell proprietary hardware. The information is collected through the SNMP interface.

The following information is modeled:

- Hardware ModelR
- Hardware Serial Number
- Operating System
- CPU Information (socket, speed, cache, voltage)
- PCI Card Information (manufacturer, model)

## 4.2. Prerequisites

### Table 4.1. Dell Hardware Prerequisites

| Prerequisite | Restriction |
|---|---|
| Product | Zenoss Core 2.2 or higher |
| Required ZenPacks | ZenPacks.zenoss.DellMonitor |
| On each remote device | The Dell OpenManage SNMP Agent is used to gather information about the device. |

## 4.3. Enable Enhanced Modeling

To enable modeling:

1. Select Infrastructure from the navigation bar.
2. Click the device name in the device list.

   The device overview page appears.
3. Select Modeler Plugins from the left panel.
4. Click Add Fields to reveal the list of available plugins.
5. Select the following plugins from the Available fields list, and then drag them to the Plugins list:
   - DellCPUMap
   - DellDeviceMap
   - DellPCIMap
6. Remove the following plugins by clicking on the 'X' button located to the right of the plugin.

- zenoss.snmp.CpuMap
- zenoss.snmp.DeviceMap

7. Click Save to save the updates.
8. Remodel the device using these new plugins by selecting Model Device from the Action menu.

# 4.4. Daemons

### Table 4.2. Daemons

| Type | Name |
|---|---|
| Modeler | **zenmodeler** |
| Performance Collector | **zenperfsnmp** |

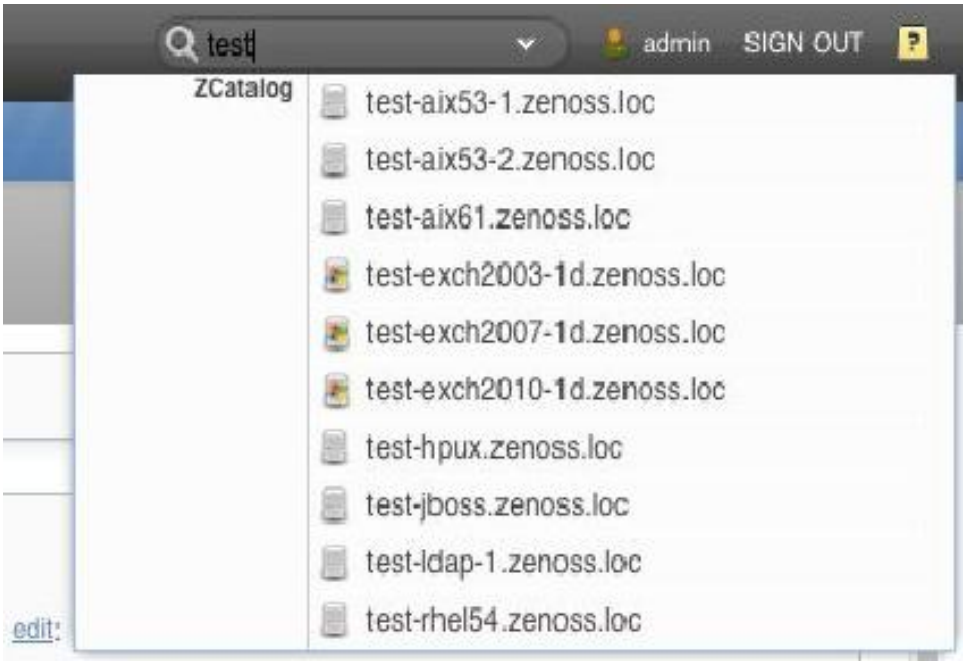
# Chapter 5. Device Search

## 5.1. About

The DeviceSearch ZenPack allows you to search, by name or IP address, for any device in the system.

A search input field appears at the top right of each page of the user interface. Typing text in this field triggers a search. You can enter partial word searches; for example, the keyword "test" matches "test123" and "test." Search results are limited to the top twenty-five matches.

**Figure 5.1. Device Search Keyword Search**




## 5.2. Prerequisites

**Table 5.1. Prerequisites**

| Prerequisite | Restriction |
|---|---|
| Product | Zenoss Core 4.2 or higher |
| Required ZenPacks | ZenPacks.zenoss.DeviceSearch |

# Chapter 6. Domain Name System (DNS)

## 6.1. About

The DigMonitor and DNSMonitor ZenPacks monitor the availability and response time of a DNS request.

## 6.2. DigMonitor

The DigMonitor ZenPack uses the check_dig Nagios plugin, which uses the `dig` command.

### 6.2.1. Prerequisites

**Table 6.1. DigMonitor Prerequisites**

| Prerequisite | Restriction |
|---|---|
| Product | Zenoss Core Version 2.2 or higher |
| Required ZenPacks | ZenPacks.zenoss.DigMonitor |

### 6.2.2. Enable Monitoring

To enable monitoring by the system:

1. Select Infrastructure from the navigation bar.
2. Click the device name in the device list.

   The device overview page appears.
3. Expand Monitoring Templates in the left panel, and then select Device.
4. Select Bind Templates from the Action menu.

   The Bind Templates dialog appears.
5. Add the DigMonitor template to the list of selected templates, and then click **OK**.

   The DigMonitor template appears under Monitoring Templates.
6. Select the DigMonitor template in the left panel, and change options as needed.

**Table 6.2. DigMonitor Data Source Options**

| Option | Description |
|---|---|
| DNS Server | Name server against which the **dig** command should be run. The default is the device host name. |
| Port | Port on which the name server is listening. This is normally port 53. |
| Record Name | Name of the record you want to look up. The default is zenoss.com. |
| Record Type | DNS record type (for example, A, MX, CNAME). |

## 6.3. DNSMonitor

The DNSMonitor ZenPack uses the check_dns Nagios plugin, which uses the `nslookup` command.

### 6.3.1. Prerequisites

#### Table 6.3. DNSMonitor Prerequisites

| Prerequisite | Restriction |
|---|---|
| Product | Zenoss Core Version 2.2 or higher |
| Required ZenPacks | ZenPacks.zenoss.DNSMonitor |

### 6.3.2. Enable Monitoring

To enable monitoring by the system:

1. Select Infrastructure from the navigation bar.
2. Click the device name in the device list.

   The device overview page appears.
3. Expand Monitoring Templates in the left panel, and then select Device.
4. Select Bind Templates from the Action menu.

   The Bind Templates dialog appears.
5. Add the DNSMonitor template to the list of selected templates, and then click **OK**.

   The DNSMonitor template appears under Monitoring Templates.
6. Select the DNSMonitor template in the left panel, and change options as needed.

#### Table 6.4. DNSMonitor Data Source Options

| Option | Description |
|---|---|
| DNS Server | Name server against which the **nslookup** command should be run. If empty (the default), the default DNS server or servers in `/etc/resolve.conf` are used. |
| Port | Port on which the name server is listening. This is normally port 53. |
| Host Name | Host name to resolve. The default is the device ID. |
| Expected IP Address | IP address to which the host name is expected to resolve. |

## 6.4. Daemons

#### Table 6.5. Daemons

| Type | Name |
|---|---|
| Performance Collector | **zencommand** |

# Chapter 7. File Transfer Protocol (FTP)

## 7.1. About

The FtpMonitor ZenPack monitors connection response time to an FTP server.

## 7.2. Prerequisites

### Table 7.1. FTP Prerequisites

| Prerequisite | Restriction |
|---|---|
| Product | Zenoss Core 2.2 or higher |
| Required ZenPacks | ZenPacks.zenoss.FtpMonitor |

## 7.3. Enable Monitoring

To enable monitoring of the device:

1. Select Infrastructure from the navigation bar.
2. Click the device name in the device list.

   The device overview page appears.
3. Expand Monitoring Templates in the left panel, and then select Device.
4. Select Bind Templates from the Action menu.

   The Bind Templates dialog appears.
5. Select the FTPMonitor template and move it to the list of selected templates.
6. Click **Save**.

   The FTPMonitor template appears under Monitoring Templates.
7. Select the FTPMonitor template and change options as needed.

### Table 7.2. FTPMonitor Basic Data Source Options

| Option | Description |
|---|---|
| Port | The port to connect to FTP server (default 21) |
| Send String | Command string to send to the server |
| Expect String | A string to expect in server response |
| Mismatch | If the expected string does not match the string returned from the remote server, create an event with one of these states: ok, warn, crit (default: warn) |
| Quit String | Command to send to the remote server to end the session |

## 7.4. Enable Secure Site Monitoring

To enable secure site monitoring:

1. Select Infrastructure from the navigation bar.
2. Click the device name in the devices list.

   The device overview page appears.
3. Expand Monitoring Templates in the left panel.
4. Select the FTPMonitor template and change options as needed.

### Table 7.3. FTPMonitor Secure Data Source Options

| Option | Description |
|---|---|
| Port | The port to connect to FTP server (default 21). |
| Certificate | Minimum days for which a certificate is valid |
| Use SSL | Use SSL for the connection |

## 7.5. Tuning for Site Responsiveness

1. Select Infrastructure from the navigation bar.
2. Click the device name in the devices list.

   The device overview page appears.
3. Expand Monitoring Templates in the left panel.
4. Select the FTPMonitor template and change options as needed.

### Table 7.4. FTPMonitor Tunables Data Source Options

| Option | Description |
|---|---|
| Timeout | Seconds before connection times out (default: 60) |
| Refuse | If a TCP/IP connection to the remote service is refused (ie no program is listening at that port) send an event with one of these severity states: ok, warn, crit (default: crit) |
| Max Bytes | Close the connection once more than this number of bytes are received. |
| Delay | Seconds to wait between sending string and polling for response |
| Warning response time (seconds) | Response time to result in a warning status. |
| Critical response time (seconds) | Response time to result in critical status |

## 7.6. Daemons

### Table 7.5. Daemons

| Type | Name |
|---|---|
| Performance Collector | **zencommand** |

# Chapter 8. HP PC Hardware

## 8.1. About

HPMonitor provides custom modeling of devices running the HP Insight Management Agents. It also contains hardware identification for HP proprietary hardware. The information is collected through the SNMP interface.

The following information is modeled:

- Hardware Model
- Hardware Serial Number
- Operating System
- CPU Information (socket, speed, cache)

## 8.2. Prerequisites

### Table 8.1. HP PC Hardware Prerequisites

| Prerequisite | Restriction |
|---|---|
| Product | Zenoss Core 2.2 or higher |
| Required ZenPacks | ZenPacks.zenoss.HPMonitor |
| On each remote device | The HP Insight SNMP Management Agent gathers information about the device. |

## 8.3. Enable Enhanced Modeling

To enable enhanced modeling:

1. Select Infrastructure from the navigation bar.
2. Click the device name in the device list.

   The device overview page appears.
3. Select Modeler Plugins from the left panel.
4. Click Add Fields to reveal the list of available plugins.
5. Select the following available plugins and drag them to the plugins list:
   - HPCpuMap
   - HPDeviceMap
6. Remove the following plugins by clicking the 'X' button to the right of the plugin:
   - zenoss.snmp.CPUMap
   - zenoss.snmp.DeviceMap

7. Click **Save**.

8. Remodel the device using the new plugins. To do this, select Model Device from the Action menu.

# 8.4. Daemons

### Table 8.2. Daemons

| Type | Name |
|---|---|
| Modeler | **zenmodeler** |
| Performance Collector | **zenperfsnmp** |

# Chapter 9. Internet Relay Chat (IRC)

## 9.1. About

ZenPacks.zenoss.IrcdMonitor monitors the number of users connected to an IRC server.

## 9.2. Prerequisites

### Table 9.1. IRC Prerequisites

| Prerequisite | Restriction |
|---|---|
| Product | Zenoss Core 2.2 or higher |
| Required ZenPacks | ZenPacks.zenoss.IRCDMonitor |

## 9.3. Enable Monitoring

To enable monitoring:

1. Select Infrastructure from the navigation bar.
2. Click the device name in the device list.

   The device overview page appears.
3. Expand Monitoring Templates in the left panel, and then select Device.
4. Select Bind Templates from the Action menu.

   The Bind Templates dialog appears.
5. Move the IrcdeMonitor template from the Available list and move it to the Selected list.
6. Click **Save**.

   The IrcdMonitor template is added.
7. Click the new template in the left panel and change options as needed.

### Table 9.2. IRC Basic Data Source Options

| Option | Description |
|---|---|
| Port | Specifies the port to connect to the IRC server (default 6667). |
| warning_num | Creates a warning event when this number of users are seen. |
| critical_num | Creates a critical event when this number of users are seen. |

## 9.4. Daemons

### Table 9.3. Daemons

| Type | Name |
|---|---|
| Performance Collector | **zencommand** |

# Chapter 10. Jabber Instant Messaging

## 10.1. About

ZenPacks.zenoss.JabberMonitor monitors the response time of devices running a Jabber server.

## 10.2. Prerequisites

### Table 10.1. Jabber Prerequisites

| Prerequisite | Restriction |
|---|---|
| Product | Zenoss Core Version 2.2 or higher |
| Required ZenPacks | ZenPacks.zenoss.JabberMonitor |

## 10.3. Enable Monitoring

To enable monitoring:

1. Select Infrastructure from the navigation bar.
2. Click the device in the device list.

   The device overview page appears.
3. Expand Monitoring Templates in the left panel, and then select Device.
4. Select Bind Templates from the Action menu.

   The Bind Templates dialog appears.
5. Move the Jabber template from the Available list to the Selected list, and then click **Save**.

   The Jabber template is added. The system can begin collecting Jabber server metrics from the device.
6. Select the newly added template and change options as needed.

### Table 10.2. Jabber Data Source Options

| Option | Description |
|---|---|
| Timeout (seconds) | Seconds before connection times out (default: 60) |
| Port | The port on which the Jabber server is listening. Typically this is port 5223. |
| Send String | string to send to the server : default<br>`<stream:stream to='${dev/id}' xmlns:stream='http://etherx.jabber.org/streams'>` |
| Expect String | String to expect in server response.<br>`<stream>` |

## 10.4. Daemons

### Table 10.3. Daemons

| Type | Name |
|---|---|
| Performance Collector | **zencommand** |

# Chapter 11. Java 2 Platform Standard Edition (J2E)

## 11.1. About

ZenJMX is a ZenPack that allows Zenoss Core to communicate with remote Java Management Extensions (JMX) agents. The ZenJMX ZenPack defines a data source named `JMX` that allows you to query any single or complex-value attribute, or invoke an MBean operation. It also comes with a built-in template named `Java` that contains MBean information for a few beans built into the JVM.

### Note

ZenJMX also includes a built-in template named `ZenJMX`. This template should be used only on devices running Java applications that make information available through JMX. To monitor other Java applications, use the included Java template.

When the **zenjmx** daemon is started it communicates with ZenHub and retrieves a list of devices that possess `JMX` data sources. It also spawns a Java process. ZenJMX asynchronously issues queries for each of those devices to the Java process via XML-RPC. The Java process then collects the data from the Java application to be monitored, and returns the results to ZenJMX. Any collection or configuration errors are sent as events to Zenoss Core and will appear in the event console.

Lastly, ZenJMX heartbeats after each collect to ZenHub to let Zenoss Core know that ZenJMX is still alive and well.

### 11.1.1. JMX Background

The JMX technology is used throughout the Java Virtual Machine to provide performance and management information to clients. Using a combination of **JConsole** (Oracle's JMX client that is shipped with the JDK) and JMX, a system operator can examine the number of threads that are active in the JVM or change the log level. There are numerous other performance metrics that can be gleaned from the JVM, as well as several management interfaces that can be invoked that change the behavior of the JVM.

In Java 5, Oracle introduced the Remote API for Java Management Extensions. This enhancement defines an RMI wrapper around a JMX agent and allows for independent client development. ZenJMX accesses remote JMX agents via the Remote API for Java Management Extensions. It currently does not support local connections (provided via the temporary directory) to JMX Agents. JMX also specifies the manner in which various protocols can be used to connect to clients, and send and receive information. The original, most commonly used protocol is RMI. ZenJMX supports RMI and JMXMP connections.

### 11.1.2. ZenJMX Capabilities

ZenJMX is a full-featured JMX client that works "out of the box" with JMX agents that have their remote APIs enabled. It supports authenticated and unauthenticated connections, and it can retrieve single-value attributes, complex-value attributes, and the results of invoking an operation. Operations with parameters are also supported so long as the parameters are primitive types (Strings, booleans, numbers), as well as the object version of primitives (such as `java.lang.Integer` and `java.lang.Float`). Multi-value responses from operations (Maps and Lists) are supported, as are primitive responses from operations.

The `JMX` data source installed by ZenJMX allows you to define the connection, authentication, and retrieval information you want to use to retrieve performance information. The IP address is extracted from the parent device, but the port number of the JMX Agent is configurable in each data source. This allows you to operate multiple JMX Agents on a

single device and retrieve performance information for each agent separately. This is commonly used on production servers that run multiple applications.

Authentication information is also associated with each JMX data source. This offers the most flexibility for site administrators because they can run some JMX agents in an open, unauthenticated fashion and others in a hardened and authenticated fashion. SSL-wrapped connections are supported by the underlying JMX Remote subsystem built into the JDK, but were not tested in the Zenoss labs. As a result, your success with SSL encrypted access to JMX Agents may vary.

The data source allows you to define the type of performance information you want to achieve: single-value attribute, complex-value attribute, or operation invocation. To specify the type of retrieval, you must specify an attribute name (and one or more data points) or provide operation information.

Any numerical value returned by a JMX agent can be retrieved by Zenoss Core and graphed and checked against thresholds. Non-numerical values (Strings and complex types) cannot be retrieved and stored by Zenoss Core.

When setting up data points, make sure you understand the semantics of the attribute name and choose the correct Zenoss Core data point type. Many JMX Agent implementations use inconsistent nomenclature when describing attributes. In some cases the term "Count" refers to an ever-increasing number (a "Counter" data point type). In other cases the term "Count" refers to a snapshot number (a "Gauge" data point type).

## 11.1.3. Allowable Parameter Types

The following primitive data types are allowed in `JMX` calls:

- `java.lang.Integer`
- `java.lang.Long`
- `java.lang.Double`
- `java.lang.Float`
- `java.lang.String`
- `java.lang.Boolean`
- `int`
- `long`
- `double`
- `float`
- `boolean`

## 11.1.4. Single Value Attribute Calls

This is the most basic usage scenario. If you are interested in retrieving a single value from an MBean in a JMX Agent, and the attribute returns simple numeric data, you fall into the "single value attribute" category. To define a single-value attribute call simply provide the fully qualified name of your MBean and then provide the name of the attribute in the Attribute Name field of the data source. Lastly, you must define a data point.

Some examples of this include the commonly referenced JDK Threading information:

- MBean Name: java.lang:type=Threading
- Attribute Name: ThreadCount

- Data Points:
  - ThreadCount (type: gauge)

Java uses lots of file descriptors during normal operation. The number of open file descriptors the JVM is working with can be measured using the following information:

- MBean Name: java.lang:type=OperatingSystem
- Attribute Name: OpenFileDescriptorCount
- Data Points:
  - OpenFileDescriptorCount (type: gauge)

There are several other single-value attributes that can be retrieved from the JDK. We recommend using **JConsole** to interactively navigate through the MBean hierarchy to determine which MBeans contain useful information to you. See Section 11.5, “Using **JConsole** to Query a JMX Agent” for additional information on how to inspect the MBeans deployed in an JMX Agent.

## 11.1.5. Complex-Value Attribute Calls

If your MBean attribute defines multiple sub-attributes (via CompositeData or Tabular) that you are interested in capturing, then you fall into the category of a "complex-value attribute" call. The JDK contains a few complex-value attributes you might be interested in capturing, including garbage collection statistics that were captured during the copy and mark-sweep compact collection cycles.

To extract data from a complex-value attribute, you must define one or more data points in the data source. The names of the data points are used as keys into the complex-value data structure returned from the MBean attribute. For JMX CompositeData attributes, the data point names are used as a key to map the results. For JMX TabularData, the data point names are used as indexes into the structure to map the result.

The JDK also provides heap memory information via a complex-value attribute. The amount of committed, used, and maximum heap memory can be viewed by setting up a complex-value attribute in Zenoss Core with the following information:

- MBean Name: java.lang:type=Memory
- Attribute Name: HeapMemoryUsage
- Data Points:
  - committed (type: gauge)
  - used (type: gauge)
  - max (type: gauge)

## 11.1.6. Example Method Calls

Some management values need to be computed. These situations frequently arise when custom MBeans are deployed alongside an enterprise application. An MBean named "Accounting" might be deployed within an enterprise application that defines operations intended for operators or support staff. These operations might include methods such as "getBankBalance()" or "countTotalDeposits()".

ZenJMX has the ability to invoke operations, but there are some subtleties in how ZenJMX sends parameters to the JMX Agent and interprets the response.

### 11.1.6.1. No parameters, single return value

In the most basic usage scenario no arguments are passed to the operation and a single value is returned. This usage scenario is very similar to a single-value attribute call, except we're invoking an operation to retrieve the value rather than accessing an attribute. The configuration for this hypothetical usage scenario follows:

- MBean Name: Application:Name=Accounting,Type=Accounting
- Operation Name: getBankBalance()
- Data Points:
  - balance (type: gauge)

### 11.1.6.2. No parameters, multiple values returned in List format

In this scenario no parameters are passed to an operation, but multiple response values are provided in a List. The values returned are expressed in a List<Object>, but they are coerced (but not casted) to doubles prior to being stored in Zenoss Core. This means that returning a numeric value as "1234" will work, but "1,234" will not work. The litmus test is to evaluate if `Double.valueOf(object.toString())` will successfully evaluate.

ZenJMX can be configured to read multiple values from an operation's results by defining multiple data points. You must define a data point for each value returned from the operation, and if there is a mismatch between the number of data points you define and the size of the List<Object> returned an exception will be generated. The configuration for ZenJMX follows:

- MBean Name: Application:Name=Accounting,Type=Accounting
- Operation Name: getBalanceSummary()
- Data Points:
  - dailyBalance (type: gauge)
  - annualBalance (type: gauge)

### 11.1.6.3. No parameters, multiple values returned in Map format

In this scenario no parameters are passed to an operation, but multiple response values are provided in a Map<String, Object>. The keyset of the Map contains the names of data points that can be defined, and the values are the values of said data points. When a Map<String, Object> is returned you need not capture all of the returned values as data points, and you can instead pick the exact values you are interested in. To choose the values to capture you simply define data points with the same names as Strings in the keyset.

The following configuration demonstrates how to extract specific data points from an operation that returns a Map<String, Object>. The key item to note in this configuration is that "dailyBalance" and "annualBalance" must be present as keys in the returned Map<String, Object> and their values must be coercible via the Double.valueOf(object.toString()) idiom.

- MBean Name: Application:Name=Accounting,Type=Accounting
- Operation Name: getBalances()
- Data Points:
  - dailyBalance (type: gauge)

- annualBalance (type: gauge)

### 11.1.6.4. Single parameter in polymorphic operation

MBeans are implemented as Java classes and Java permits parameterized polymorphic behavior. This means that multiple methods can be defined with the same name so long as their parameter signatures differ. You can safely define "getBalance(String)" and "getBalance()" and the two exist as separate methods.

In order to properly resolve methods with the same name the caller must provide a Class[] that lists the types of parameters that exist in the method's signature. This resolves the candidate methods to an individual method which can then be invoked by passing an Object[].

ZenJMX allows you to resolve methods of the same name and asks you to provide the fully qualified class names of each parameter in comma delimited format when you set up the data source. Note that primitive types (String, Boolean, Integer, Float) are supported but complex types are not supported, and that you must include the class' package name when providing the information (java.lang.String).

The Object[] of parameter values must line up with Class[] of parameter types, and if there is a mismatch in the number of types and values that are provided an exception will be generated.

The marshaling of values from String to Boolean, Integer, and Float types is provided via the .valueOf() static method on each of those types. That is, if you define an attribute of type java.lang.Integer you must provide a String that can be successfully passed to java.lang.Integer.fromValue(). If you fail to do so an exception is generated.

This example illustrates how to pass a single parameter to a polymorphic operation:

- MBean Name: Application:Name=Accounting,Type=Accounting
- Operation Name: getBalances()
- Paramater Types: java.lang.Integer
- Parameter Values: 1234
- Data Points:
  - balance (type: gauge)

Here is another example where we've changed the type of the parameter passed to the method to be a String. Semantically it represents a different type of Account in our example:

- MBean Name: Application:Name=Accounting,Type=Accounting
- Operation Name: getBalances()
- Paramater Types: java.lang.String
- Parameter Values: sbb552349999
- Data Points:
  - balance (type: gauge)

### 11.1.6.5. Multiple parameters in polymorphic operations

The above example describes how polymorphic behavior in Java functions and how method resolution can be provided by identifying the Class[] that represents the parameters passed to a method. The situation where multiple parameters

are passed to a polymorphic operation is no different then the situation where a single parameter is passed to a polymorphic operation, except that the length of the Class[] and Object[] is greater than one.

When multiple parameters are required to invoke an operation you must provide the fully qualified class names of each parameter's type in comma delimited format, as well as the object values for each type (also in comma delimited format).

The following example demonstrates a configuration that passes two parameters to an MBean operation. The second parameter passed is a default value to return if no account can be located matching the first parameter.

- MBean Name: Application:Name=Accounting,Type=Accounting
- Operation Name: getBalances()
- Parameter Types: java.lang.String, java.lang.Integer
- Parameter Values: sbb552349999, 0
- Data Points:
  - balance (type: gauge)

There are additional combinations that are possible with polymorphic methods and the values they return, and those combinations are left as an exercise for the reader to explore. The logic for extracting results from multi-value operation invocations follows the same rules as the logic for extracting results from a multi-value attribute read. For additional information on the rules of that logic see the section above on multi-value attributes.

### 11.1.7. Special Service URLs

By default, URLs are assembled as:

```
service:jmx:rmi:///jndi/rmi://hostName:portNum/jmxrmi
```

This host name and port points to a registry. After a JMX agent connects to the registry, the registry tells the agent which host and port to use for remote calls.

In some situations, you may want to explicitly provide the registry host and port, as well as the host and port for the remote calls. Use the long form, as in:

```
service:jmx:rmi://127.0.0.1:8999/jndi/rmi://127.0.0.1:8999/jmxrmi
```

## 11.2. Prerequisites

**Table 11.1. J2EE Prerequisites**

| Prerequisite | Restriction |
|---|---|
| Product | Zenoss Core Version 2.2 or higher |
| Required ZenPacks | ZenPacks.zenoss.ZenJMX |
| Other | Oracle JRE Version 5.0 or higher |

### 11.2.1. Oracle Java Runtime Environment (JRE)

ZenJMX requires Oracle JRE Version 5.0 or higher. Make sure that after you install the JRE you update your PATH such that the **java** executable works. You can test this using the command:

```
$ which java
/usr/java/default/bin/java
```

If the above returns a fully qualified path, then you have successfully installed Java.

If Java is not installed, the **which** will return a message similar to the following:

```
$ which java
/usr/bin/which: no java in (/usr/local/bin:/bin:/usr/bin:/opt/zenoss/bin)
```

To determine which version of Java is installed, run the following command:

```
$ java -version
java version "1.5.0_16"
Java(TM) 2 Runtime Environment, Standard Edition (build 1.5.0_16-b06-284)
Java HotSpot(TM) Client VM (build 1.5.0_16-133, mixed mode, sharing)
```

### Warning

Oracle's Java Version 5 (1.5) **must** be installed. The GNU Java does not work.

# 11.3. Example to Monitor a JMX Value

## 11.3.1. Enabling Remote JMX Access

Each application server has a slightly different process for enabling remote JMX Access. You should consult with your application server for specific instructions. This section includes instructions for a few commonly used configurations.

JMX agents can be configured in two ways: remote access and local-only. When configured for remote access a JMX client communicates with the JMX agent via a socket and uses a remote protocol such as Remote Method Invocation (RMI) or JMXMP to access the MBeans. When configured for local-only access the JMX agent periodically dumps serialized MBeans to a temporary directory on the machine. **JConsole** can be used to access JMX agents in local-only mode as well as in remote mode. ZenJMX can be used only with remote servers via RMI or JMXMP and cannot work with local-only serialized MBeans. This is not a significant limitation because ZenJMX can establish RMI connections to localhost in the same manner that it creates connections to remote hosts.

The `JAVA_OPTS` environment variable can be used to enable remote access to JVM MBeans. Set it as follows:

```
JAVA_OPTS="-Dcom.sun.management.jmxremote.port=12345
JAVA_OPTS="${JAVA_OPTS} -Dcom.sun.management.jmxremote.authenticate=false"
JAVA_OPTS="${JAVA_OPTS} -Dcom.sun.management.jmxremote.ssl=false"

export JAVA_OPTS
```

When starting an application pass the `JAVA_OPTS` variable as an argument to the JVM as follows:

```
java ${JAVA_OPTS} -classpath /path/to/application.jar com.yourcompany.Main
```

You can then use **JConsole** to connect to localhost:12345. Authentication can be configured by modifying the java.security file as well as java.policy. There are lots of examples available on the Internet that can provide guidance in how to achieve authenticated remote access to JVM MBeans.

## 11.3.2. Configure Zenoss Core with a Custom Data Source

Custom JMX data sources allow system administrators to monitor any attribute or operation result accessible via a JMX call. ZenJMX creates a `JMX` data source and allows you to provide object information, as well as authentication settings,

and attribute/operation information. Determining which object and attribute names, as well as which operations to invoke, is the key to customizing ZenJMX.

To configure the system with a custom data source:

1. Select Infrastructure from the navigation bar.
2. Click the device in the device list.

   The device overview page appears.
3. Expand Monitoring Templates in the left panel, and then select Device.
4. Select Add Local Template from the Action menu.

   The Add Local Template dialog appears.
5. Enter a name for the template (such as JVM Values), and then click **Submit**.

   The template is added.
6. Select the newly created template.
7. Click  (Add) in the Data Sources area.

   The Add Data Source dialog appears.
8. Enter a name for the data source (Heap Memory), select JMX as the type, and then click Submit.

   The data source is added.
9. Double-click the data source to edit it. Change options as needed, and then click **Save**.

### Table 11.2. Memory Head Example ZenJMX Data Source Options

| Option | Description |
|---|---|
| Protocol | RMI or JMXMP. Consult your Java application documentation to determine which JMX Connector protocols it supports. |
| JMX Management Port | This is not necessarily the same as the listen port for your server. |
| Object Name | The Object Name is also referred to as the MBean name. Enter `java.lang:type=Memory` |
| Attribute Name | Enter `HeapMemoryUsage` |

10. Add data points named `committed`, `max`, and `used`:

    a. Select Add Data Point from the Action menu.

       The Add Data Point dialog appears.

    b. Enter the name of the data point (`committed`, `max`, or `used`) and then click **Submit**.
11. After adding all data points, add graphs that reference them. (For more information, see *Zenoss Core Administration*.)

Review Section 11.5, “Using **JConsole** to Query a JMX Agent” to learn how to determine the object name, attribute name, and data points that might be interesting in your application.

# 11.4. Monitor Values in TabularData and CompositeData Objects

The Attribute Path input value on the ZenJMX data source allows you to monitor values nested in the TabularData and CompositeData complex open data objects. Using this value you can specify a path to traverse and index into these complex data structures.

If the result of traversing and extracting a value out of the nested open data is a single numeric value then it is automatically mapped to the datapoint in the data source. However, if the value from the open data is another open data object then the data point names from the datasource are used as indexes or keys to map values out of the open data.

The input value is a dot-separated string that represents a path through the object. Non-bracketed values are keys into CompositeData. Bracketed values are indexes into TabularData.

For TabularData indexes with more than one value, use a comma-separated list with no spaces (for example, [key1,key2]).

To specify a column name (needed only when the table has more than two columns) use curly brackets after the table index.

## Example

To get the used Tenured Generation memory after the last garbage collection from the Garbage Collector MBean, set the Attribute Name on the datasource to lastGcInfo. Set the Attribute Path to:

```
memoryUsageAfterGc.[Tenured Gen].{value}.used
```

The key `memoryUsageAfterGc` is evaluated against the CompositeData returned from the `lastGcInfo` attribute. The evaluation results in a TabularData object. Then, the `[Tenured Gen]` index is evaluated against the TableData, which returns a row in the table.

Since a row in the table can contain multiple columns, the key `value` (in curly brackets) is used to pick a column in the row. Lastly, the key `used` is evaluated against the CompositeData in the column to return the memory value.

In this example, since the index being used for the tabular data is not a multi-value index and so the column name is optional. The Attribute Path can be written as:

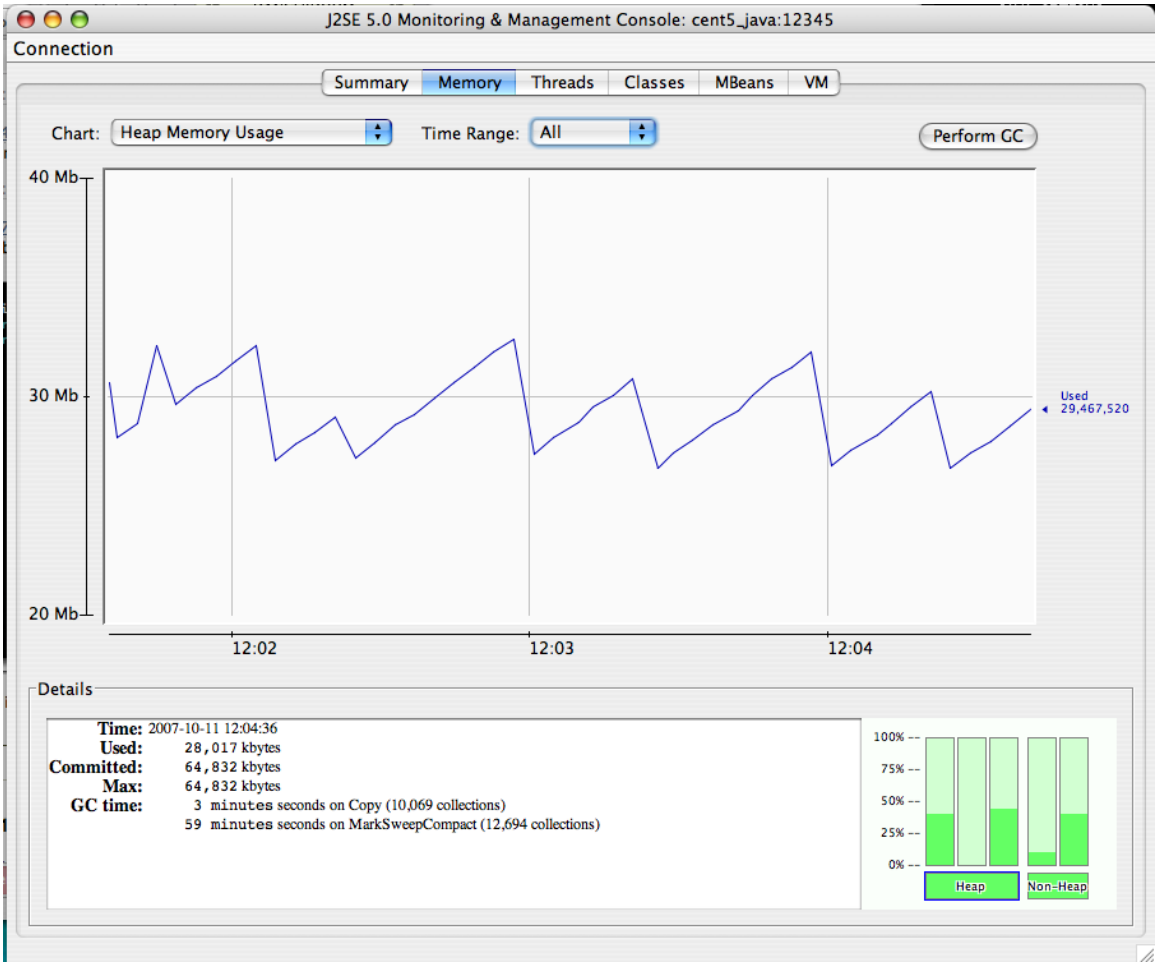
```
memoryUsageAfterGc.[Tenured Gen].used
```

# 11.5. Using JConsole to Query a JMX Agent

**JConsole** is a tool built into the JDK that allows system administrators to query a JMX Agent and examine the MBeans deployed within the server. **JConsole** also allows administrators to view JVM summary information, including the amount of time the JVM has been running, how many threads are active, how much memory is currently used by the heap, how many classes are currently loaded, and how much physical memory exists on the machine.

**JConsole** also provides a graph that shows memory, thread, and class usage over time. The scale of the graph can be adjusted so that a system administrator can examine a specific period of time, or can zoom out to view a longer range picture of usage. Unfortunately, **JConsole** can only produce graphs that show usage while **JConsole** was running. Administrators cannot look back in time to a point where the JVM was running but **JConsole** was not monitoring the JVM.

**Figure 11.1. JMX Heap Graph**

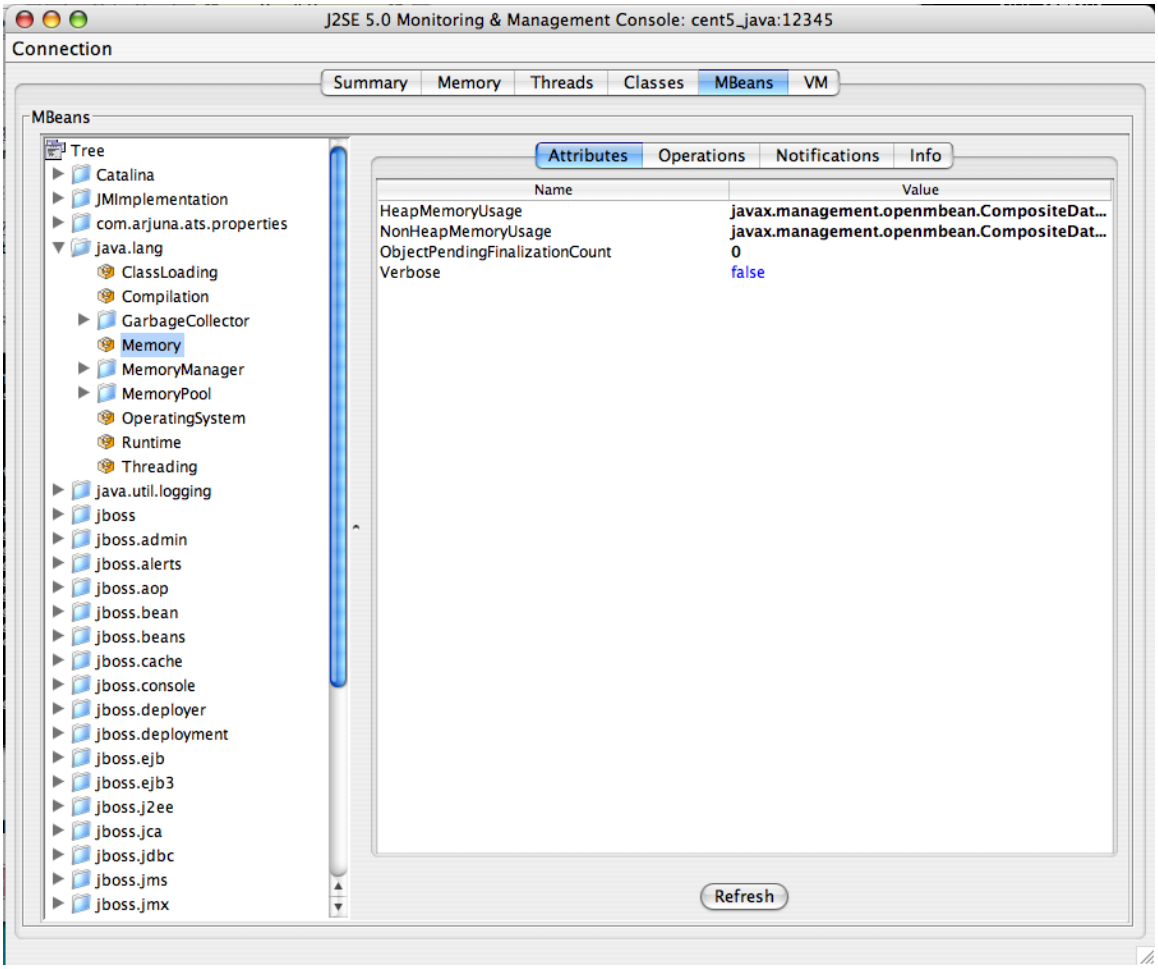



The MBeans tab along the top of **JConsole** provides an interactive method for examining MBean values. After clicking on the MBeans tab a panel will be displayed with a tree on the left hand side. The tree contains a hierarchical list of all MBeans deployed in the JVM.

The standard JVM MBeans are all in the java.lang and java.util.logging packages. Application server specific MBeans do not follow any standard naming pattern. Some vendors choose to use package names for their MBean names while other vendors choose package-like names (but not fully qualified packages).

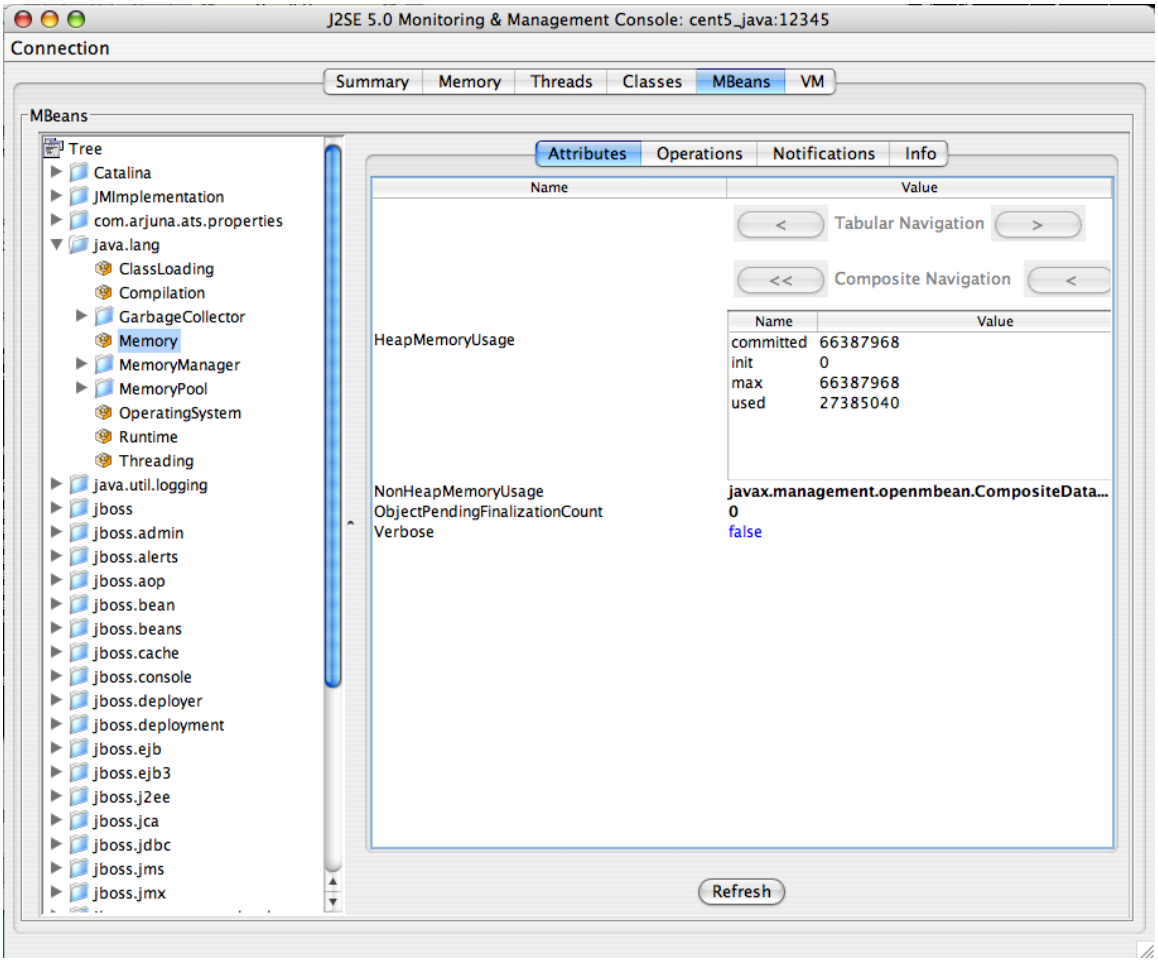
To get started expand the java.lang node in the Tree. This will expose several MBeans as well as additional folders. Click on the Memory MBean and observe how the right hand side of the panel is populated with information about the Memory MBean.

**Figure 11.2. Memory MBean**




MBeans can contain attributes and operations. MBeans can also fire notifications to observers, but that's beyond the scope of this document. The attributes tab lists all of the attributes in the first column and their values (or a clickable attribute type) in the second column. In the case of Memory the HeapMemoryUsage is a Composite attribute, otherwise referred to as a "complex-value attribute" in Zenoss Core. Double click the "javax.management.openmbean.CompositeDataSupport" type and you will see multiple attributes appear. The show the amount of committed, maximum, and used memory sizes for the heap.

**Figure 11.3. Memory MBean Expanded**



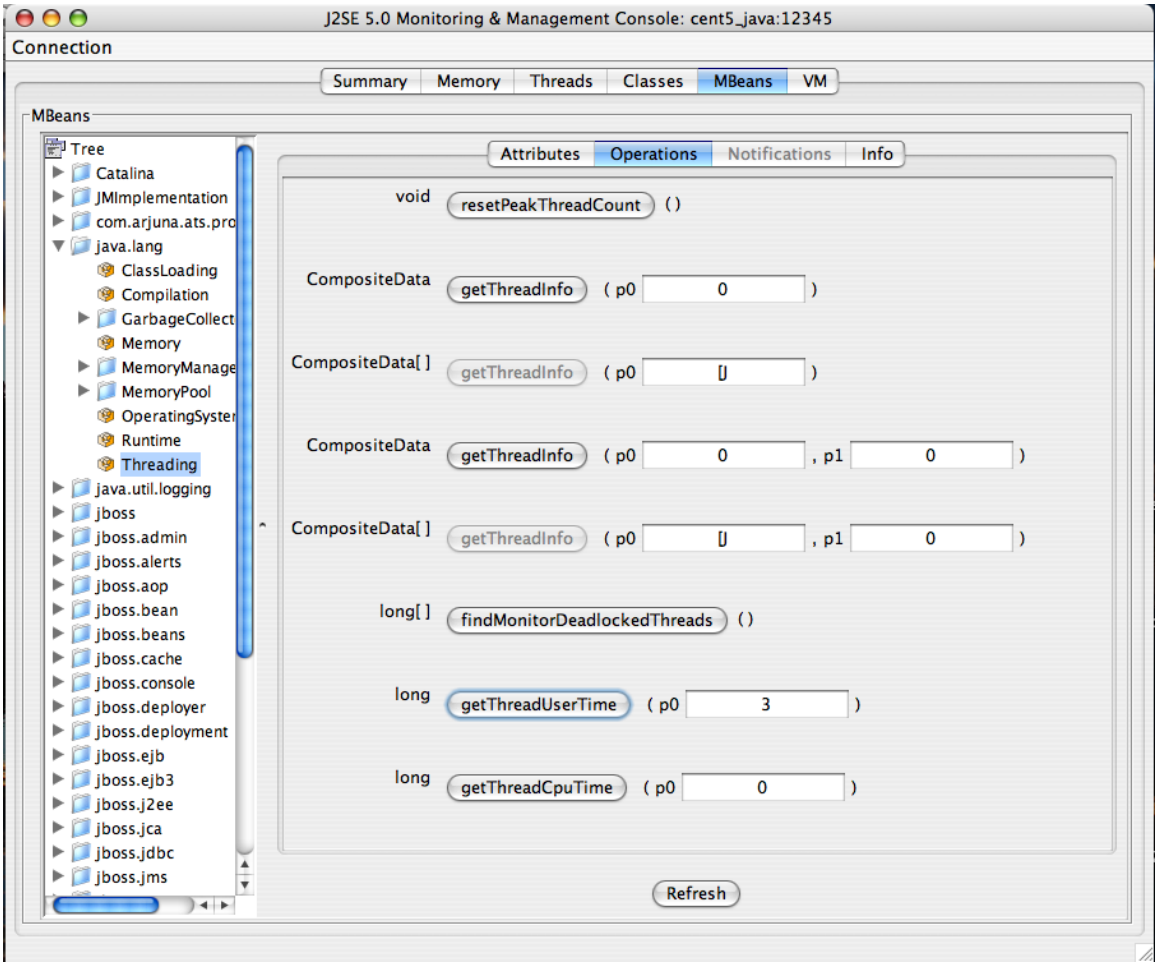

The unique name of the MBean can be viewed by clicking on the Info tab. The first value is MBean Name. Its value in the case of Memory is: "java.lang:type=Memory."

## Note

There is no standardized way to name MBeans; application server vendors name them differently.

You can also examine operation information by clicking on the Operations tab. These are methods that **JConsole** can remotely invoke on an MBean that will result in some value being computed or some state changing in the application. The Threading MBean has several operations that can be invoked that return information. Click on the java.lang package and then click on the Threading operation. Lastly, click on the Operations tab. Methods like "getThreadUserTime" are invocable.

**Figure 11.4. Operations Tab**




Test the "getThreadUserTime" method by changing the p0 parameter to 1 and clicking the "getThreadUserTime" button. A dialog window will be raised that displays the amount of CPU user time thread #1 has used. Try adjusting the parameter to different values to observe the different CPU times for the threads.

# 11.6. ZenJMX Options

Run the following command forZenJMX options:

```
zenjmx help
```

# 11.7. Memory Allocation

Use the --javaheap option to set the max heap. By default, the memory allocated is 512MB.

# 11.8. ZenJMX Logging

You can adjust logging levels to reduce the size of ZenJMX log files. In the `log4j.properties` file (in `$ZENHOME/Products/ZenJMX`), update the first line and change DEBUG to INFO, WARN, or ERROR.

# 11.9. Daemons

### Table 11.3. Daemons

| Type | Name |
|---|---|
| Performance Collector | **zenjmx** |

# Chapter 12. Lightweight Directory Access Protocol (LDAP) Response Time

## 12.1. About

ZenPacks.zenoss.LDAPMonitor monitors the response time of an LDAP server (in milliseconds).

## 12.2. Prerequisites

**Table 12.1. LDAP Monitoring Prerequisites**

| Prerequisite | Restriction |
|---|---|
| Product | Zenoss Core Version 2.2 or higher |
| Required ZenPacks | ZenPacks.zenoss.LDAPMonitor |

## 12.3. Enable Monitoring

The LDAPServer template must be bound to the device class or device you want to monitor.

### 12.3.1. For a Device

To enable monitoring for a device:

1. Select Infrastructure from the navigation bar.
2. Click the device name in the device list.

   The device overview page appears.
3. Select Configuration Properties from the left panel.
4. Modify configuration property values as needed for your environment. Check with your LDAP administration for more information.

   **Table 12.2. LDAPServer Configuration Properties**

   | Property | Description |
   |---|---|
   | `zLDAPBaseDN` | The Base Distinguished Name for your LDAP server. Typically this is the organization's domain name (for example, `dc=foobar,dc=com`) |
   | `zLDAPBindDN` | The Distinguished Name to use for binding to the LDAP server, if authentication is required |
   | `zLDAPBindPassword` | The password to use for binding to the LDAP server, if authentication is required |
5. Click **Save**.

6. Expand Monitoring Templates, and then select Device from the left panel.
7. Select Bind Templates from the Action menu.

   The Bind Templates dialog appears.
8. Add the LDAPServer template to the list of selected templates, and then click **Submit**.

   The LDAPServer template is added to the list of monitoring templates.
9. Select the LDAPServer template and change options as needed.

### Table 12.3. LDAPServer Basic Data Source Options

| Option | Description |
|---|---|
| Port | The port to connect to LDAP server (default 389) |
| Base Distinguished Name | Defaults to `${here/zLDAPBaseDN}` |
| Bind Password | Defaults to `${here/zLDAPBindPassword}` |
| Use SSL | Use SSL for the connection |

#### Note

If your LDAP servers require SSL or a custom port, select the ldap data source, and then change the Use SSL and Port fields as needed.

10. Validate your configuration by running zencommand and observing that the check_ldap or check_ldaps command correctly connects to your LDAP server:

```
zencommand run -v10 -d yourdevicenamehere
```

## 12.4. Daemons

### Table 12.4. Daemons

| Type | Name |
|---|---|
| Performance Collector | **zencommand** |

# Chapter 13. MySQL Database

## 13.1. About

MySqlMonitor provides a method for pulling performance metrics from the MySQL database server directly into Zenoss Core without requiring the use of an agent. This is accomplished by using the MySQL client library to connect to the database remotely.

The following metrics are collected and graphed for MySQL server:

- Command Statistics (SELECT, INSERT, UPDATE, DELETE)
- Select Statistics (Scan, Range Check, Range Join, Full Join)
- Handler Statistics (Keyed and Unkeyed Reads, Writes, Updates, Deletes)
- Network Traffic (Received and Sent)

## 13.2. Prerequisites

**Table 13.1. MySQL Prerequisites**

| Prerequisite | Restriction |
|---|---|
| Product | Zenoss Core 2.2 or higher |
| Required ZenPacks | ZenPacks.zenoss.MySqlMonitor |
| MySQL client | Each remote collector must have an installed MySQL client |

## 13.3. Enable Monitoring

Use the following procedures to enable monitoring.

### 13.3.1. Authorize MySQL Performance Data Access

Follow these steps to set up your MySQL server to allow Zenoss Core to read performance data from the system tables.

1. Connect to the MySQL database by using the MySQL client:

   ```
   mysql -u root
   ```

   Alternatively, if there is a MySQL root password:

   ```
   mysql -u root -p
   ```

2. Create a user for Zenoss Core to use.

   ```
   mysql> CREATE USER Name IDENTIFIED BY 'ZenossCorePassword';

   Query OK, 0 rows affected (0.00 sec)
   ```

### 13.3.2. Set up Zenoss Core

1. Select Infrastructure from the navigation bar.

2. Click the device name in the device list.

   The device overview page appears.

3. Select Configuration Properties from the left panel.

4. Edit the zMySqlRootPassword configuration property for the device or devices in Zenoss Core on which you want to monitor MySQL.

5. Click **Save**.

6. Expand Monitoring Templates, and then select Device from the left panel.

7. Select Bind Templates from the Action menu.

   The Bind Templates dialog appears.

8. Add the MySQL template to the list of selected templates, and then click **Submit**.

   The MySQL template is added to the list of monitoring templates.

### Note

Pay particular attention to the MySQL Version 5+ setting in the data source. If you are monitoring pre-v5 installations of MySQL, then you must set this value to False. If you are monitoring pre-v5 and v5+ installations, then create two templates: one for MySQL installations earlier than v5 and another for those after.

## 13.4. Daemons

**Table 13.2. Daemons**

| Type | Name |
|---|---|
| Performance Collector | **zencommand** |

# Chapter 14. Network News Transport Protocol (NNTP)

## 14.1. About

ZenPacks.zenoss.NNTPMonitor ZenPack monitors the response time of an NNTP server in milliseconds.

## 14.2. Prerequisites

### Table 14.1. NNTP Prerequisites

| Prerequisite | Restriction |
|---|---|
| Product | Zenoss Core Version 2.2 or higher |
| Required ZenPacks | ZenPacks.zenoss.NNTPMonitor |

## 14.3. Enable Monitoring

To enable monitoring for a device:

1. Select Infrastructure from the navigation bar.
2. Click the device name in the device list.

   The device overview page appears.
3. Expand Monitoring Templates, and then select Device from the left panel.
4. Select Bind Templates from the Action menu.

   The Bind Templates dialog appears.
5. Add the NNTPMonitor template to the list of selected templates, and then click **Submit**.

   The NNTPMonitor template is added to the list of monitoring templates.
6. Select the template and change options as needed.
7. Validate your configuration by running **zencommand** and observing that the **check_nntp** or **check_nntps** command correctly connects to your NNTP server:

   ```
   zencommand run -v10 -d yourdevicenamehere
   ```

## 14.4. Daemons

### Table 14.2. Daemons

| Type | Name |
|---|---|
| Performance Collector | **zencommand** |

# Chapter 15. Network Time Protocol (NTP)

## 15.1. About

ZenPacks.zenoss.NtpMonitor monitors the offset between system time and a target NTP (Network Time Server) server's time.

## 15.2. Prerequisites

### Table 15.1. NTP Prerequisites

| Prerequisite | Restriction |
|---|---|
| Product | Zenoss Core Version 2.2 or higher |
| Required ZenPacks | ZenPacks.zenoss.NtpMonitor |

## 15.3. Enable Monitoring

The NTPMonitor template must be bound to the device class or device you want to monitor.

1. Select Infrastructure from the navigation bar.
2. Click the device name in the device list.

   The device overview page appears.
3. Expand Monitoring Templates, and then select Device from the left panel.
4. Select Bind Templates from the Action menu.

   The Bind Templates dialog appears.
5. Add the NTPMonitor template to the list of selected templates, and then click **Submit**.

   The NTPMonitor template is added to the list of monitoring templates. You can now start collecting the NTP server metrics from this device.

## 15.4. Daemons

### Table 15.2. Daemons

| Type | Name |
|---|---|
| Performance Collector | **zencommand** |

# Chapter 16. ONC-Style Remote Procedure Call (RPC)

## 16.1. About

ZenPacks.zenoss.RPCMonitor monitors the availability of an ONC RPC server.

## 16.2. Prerequisites

### Table 16.1. ONC RPC Prerequisites

| Prerequisite | Restriction |
|---|---|
| Product | Zenoss Core Version 2.2 or higher |
| Required ZenPacks | ZenPacks.zenoss.RPCMonitor |

## 16.3. Enable Monitoring

The RPCMonitor template must be bound to the device class or device you want to monitor. Follow these steps to enable monitoring:

1. Select Infrastructure from the navigation bar.
2. Click the device name in the device list.

   The device overview page appears.
3. Select Configuration Properties from the left panel.
4. Set the appropriate RPC command to test in the zRPCCommand configuration property (for example, nfs or ypserv).
5. Click **Save**.
6. Expand Monitoring Templates, and then select Device from the left panel.
7. Select Bind Templates from the Action menu.

   The Bind Templates dialog appears.
8. Add the RPCServer template to the list of selected templates, and then click **Submit**.

   The RPCServer template is added to the lists of monitoring templates. You can now collect the RPCServer server metrics from the device.

## 16.4. Daemons

### Table 16.2. Daemons

| Type | Name |
|---|---|
| Performance Collector | **zencommand** |

# Chapter 17. SSH Monitoring Example

## 17.1. About

The LinuxMonitor ZenPack demonstrates the new Secure Shell (SSH) features. This example ZenPack includes functionality to model and monitor several types of device components for devices placed in the `/Server/SSH/Linux` device class by running commands and parsing the output. Parsing of command output is performed on the Zenoss Core server or on a distributed collector. The account used to monitor the device does not require root access or special privileges.

This ZenPack is provided for developers as it provides some examples of how to create SSH performance collecting plugins. See the *Zenoss Developer's Guide* for more information about additional SSH features.

## 17.2. Prerequisites

**Table 17.1. Linux SSH Monitoring Example Prerequisites**

| Prerequisite | Restriction |
|---|---|
| Product | Zenoss Core Version 2.4 or higher |
| Required ZenPacks | ZenPacks.zenoss.LinuxMonitor |

## 17.3. Set Linux Server Monitoring Credentials

All Linux servers must have a device entry in an organizer below the `/Devices/Server/SSH/Linux` device class.

### Tip

The SSH monitoring feature will attempt to use key-based authentication before using a configuration properties password value.

1. Select Infrastructure from the navigation bar.
2. Click the device name in the device list.

   The device overview page appears.
3. Select Configuration Properties from the left panel.
4. Verify the credentials for the service account.

**Table 17.2. Linux Configuration Properties**

| Name | Description |
|---|---|
| zCommandUsername | Linux user with privileges to gather performance information. |
| zCommandPassword | Password for the Linux user. |

## 17.4. Add a Linux Server

The following procedure assumes that credentials have been set.

1. Select Infrastructure from the navigation bar.
2. Select Add a Single Device from the Add Device list of options.

   The Add a Single Device dialog appears.
3. Enter the following information in the dialog:

### Table 17.3. Adding Linux Device Details

| Name | Description |
|---|---|
| Name or IP | Linux host to model. |
| Device Class | `/Server/SSH/Linux` |
| Model Device | Select this option unless adding a device with a user name and password different than found in the device class. If you do not select this option, then you must add the credentials (see Section 17.3, “Set Linux Server Monitoring Credentials”) and then manually model the device. |

4. Click **Add**.

## 17.5. Daemons

### Table 17.4. Daemons

| Type | Name |
|---|---|
| Modeler | **zenmodeler** |
| Performance Collector | **zencommand** |

# Chapter 18. VMware esxtop

## 18.1. About

The EsxTop ZenPack uses the `resxtop` command to gather performance information about VMware Infrastructure™ ESX™ servers. A basic modeler creates virtual machines under the `/Devices/Server/Virtual Hosts/EsxTop` device class for any host device that is added and modeled.

## 18.2. Prerequisites

To implement this ZenPack, you must:

- Install the OpenSSL development package, Version 0.9.7 or higher
- Install the VMware vSphere CLI (as described in the section titled Installing Prerequisite Libraries).
- Update the ZenossVirtualHostMonitor ZenPack to Version 2.3.5.

### Table 18.1. Prerequisites

| Prerequisite | Restriction |
|---|---|
| Product | Zenoss Core Version 3.0 or higher |
| Required ZenPacks | ZenPacks.zenoss.ZenossVirtualHostMonitor<br><br>ZenPacks.zenoss.EsxTop |
| VMware vSphere™ Command-Line Interface (CLI) | VMware vSphere CLI Version 4.1 or higher must be installed on the Zenoss Core collectors. |

### 18.2.1. Installing Prerequisite Libraries

The VMware vSphere CLI is required for access to the `resxtop` command, which enables Zenoss Core to model and gather performance information about individual ESX servers.

Follow these steps to install the CLI and required software:

1. If you have not yet installed it, install the OpenSSL development package. For example, for an RPM-based system, enter:

   ```
   yum install openssl-devel
   ```

2. From your VMware account, download the VMware vSphere CLI.

   #### Note

   For downloads and documentation, go to:

   http://downloads.vmware.com/d/details/vcli41/ZHcqYmRoaCpiZHRAag==

3. Copy the package to each Zenoss Core collector.
4. For each collector:

a. Expand the package file.

b. Run the following command to install the package:

```
./vmware-install.pl
```

c. As the zenoss user, run the following command to verify successful installation:

```
resxtop --server myESXServer --user userOnRemoteEsxServerAllowedToUseEsxTop -b -n 1 -a
```

The `resxtop` command prompts for a password.

d. Enter the password for a user with permissions on the remote ESX server.

If the command is working correctly, then a screen displays with several pages of command output.

e. Create a symbolic link from the location that the `resxtop` command was installed into the `$ZENHOME/libexec` directory. This allows the `check_esxtop` command to automatically determine which binary to run. For example:

```
cd $ZENHOME/libexec
ln -s PathToResxtop
```

f. Test the `check_esxtop` command by showing the VMs on the remote server:

```
$ZENHOME/ZenPacks/Ze*EsxTop*/Z*/z*/E*/libexec/check_esxtop --server=myEsxserver \
 --user=userOnRemoteEsxServerAllowedToUseEsxTop --password=password --showvms
```

# 18.3. Enabling the ZenPack

Follow these steps to set up the EsxTop ZenPack. From the Zenoss Core interface, add a host:

1. From Infrastructure > Devices, navigate to the `/Devices/Server/Virtual Hosts/EsxTop` device class.
2. From  , select Add a Single Device.

   The Add a Single Device dialog appears.
3. Enter a host name or IP address.
4. De-select the Model Device option.
5. Click **Add**.
6. Select the newly added device in the list.

   The device overview appears.
7. Click **Details**, and then select Configuration Properties in the left panel.
8. Enter login credentials for the zCommandUsername and zCommandPassword configuration properties, and then click **Save**.
9. If the device has an SNMP agent installed, update the ESX device configuration with the appropriate SNMP configuration information, and then add any desired modeler plugins.
10. From  (Action menu), select Model device.

# 18.4. Daemons

### Table 18.2. Daemons

| Type | Name |
|---|---|
| Modeler | **zenmodeler** |
| Performance Collector | **zencommand** |

# Chapter 19. Web Page Response Time (HTTP)

## 19.1. About

ZenPacks.zenoss.HttpMonitor monitors connection response time to an HTTP server and determines whether specific content exists on a Web page.

## 19.2. Prerequisites

**Table 19.1. HTTP Prerequisites**

| Prerequisite | Restriction |
|---|---|
| Product | Zenoss Core Version 2.2 or higher |
| Required ZenPacks | ZenPacks.zenoss.HttpMonitor |

## 19.3. Enable Monitoring

Follow these steps to enable monitoring:

1. Select Infrastructure from the navigation bar.
2. Click the device name in the device list.

   The device overview page appears.
3. Expand Monitoring Templates, and then select Device from the left panel.
4. Select Bind Templates from the Action menu.

   The Bind Templates dialog appears.
5. Add the HttpMonitor template to the list of selected templates, and then click **Submit**.

   ### Note

   Prior to Zenoss Core 2.4, this template was not available. If you are using an earlier release, you must create the template, data source and graphs manually.
6. The HttpMonitor template is added to the list of monitoring templates. You can now begin collecting Web server metrics from the device.

## 19.4. Check for a Specific URL or Specify Security Settings

1. Select Infrastructure from the navigation bar.
2. Click the device name in the device list.

   The device overview page appears.

3. Expand Monitoring Templates, and then select Device from the left panel.
4. Create a local copy of the template.
5. Select the newly created local template copy.
6. Select the HttpMonitor data source, and then select View and Edit Details from the Action menu.

   The Edit Data Source dialog appears.
7. Change data source options as needed, and then click **Save**.

### Table 19.2. HTTPMonitor Content Checking Data Source Options

| Option | Description |
|---|---|
| Port | The port to connect to HTTP server (default 80). |
| Use SSL | Use SSL for the connection |
| Url | Address of the web page. |
| Basic Auth User | If the website requires credentials, specify the username here. |
| Basic Auth Password | Password for the user. |
| Redirect Behavior | If the web site returns an HTTP redirect, should the probe follow the redirect or create an event? Possible event severities are `OK`, `Warning`, and `Critical`. |

## 19.5. Check for Specific Content on the Web Page

This procedure allows Zenoss Core to create an event if content at the web page does not match the expected output.

1. Select Infrastructure from the navigation bar.
2. Click the device name in the device list.

   The device overview page appears.
3. Expand Monitoring Templates, and then select Device from the left panel.
4. Create a local copy of the template.
5. Select the newly created local template copy.
6. Select the HttpMonitor data source, and then select View and Edit Details from the Action menu.

   The Edit Data Source dialog appears.
7. Change data source options as needed, and then click **Save**.

### Table 19.3. HTTPMonitor Content Checking Data Source Options

| Option | Description |
|---|---|
| Regular Expression | A Python regular expression to match text in the web page. |
| Case Sensitive | Is the regular expression case-sensitive or not? |
| Invert Expression | If you would like to test to see if the web page does **not** contain content matched by a regular expression, check this box. |

# 19.6. Tuning for Site Responsiveness

1. Select Infrastructure from the navigation bar.
2. Click the device name in the device list.

   The device overview page appears.
3. Expand Monitoring Templates, and then select Device from the left panel.
4. Create a local copy of the template.
5. Select the newly created local template copy.
6. Select the HttpMonitor data source, and then select View and Edit Details from the Action menu.

   The Edit Data Source dialog appears.
7. Change data source options as needed, and then click **Save**.

### Table 19.4. HTTPMonitor Tunables Data Source Options

| Option | Description |
|---|---|
| Timeout (seconds) | Seconds before connection times out (default: 60) |
| Cycle Time (seconds) | Number of seconds between collection cycles (default: 300 or five minutes) |

# 19.7. Daemons

### Table 19.5. Daemons

| Type | Name |
|---|---|
| Performance Collector | **zencommand** |

# Chapter 20. Xen Virtual Hosts

## 20.1. About

The XenMonitor ZenPack allows you to monitor Xen para-virtualized domains with Zenoss Core.

This ZenPack:

- Extends ZenModeler to discover guests running on the Xen host.
- Provides screens and templates for collecting and displaying resources allocated to guests.

The XenMonitor ZenPack requires the ZenossVirtualHostMonitor ZenPack to be installed as a prerequisite.

## 20.2. Prerequisites

### Table 20.1. Xen Virtual Hosts Prerequisites

| Prerequisite | Restriction |
|---|---|
| Product | Zenoss Core Version 2.2 or higher |
| Required ZenPacks | ZenPacks.zenoss.XenMonitor<br>ZenPacks.zenoss.ZenossVirtualHostMonitor |

## 20.3. Model Hosts and Guest

For each Xen server, follow this procedure:

1. Optionally, place an SSH key to your Xen server to allow the zenoss user from the Zenoss Core server to log in as root without requiring further credentials.
2. Create the Xen server in the `/Servers/Virtual Hosts/Xen` device class.

   ### Warning

   If you have this server modeled already, remove the server and recreate it under the Xen device class. Do not move it.

3. Select the Guest menu and ensure that the guest hosts were found during the modeling process.

## 20.4. Daemons

### Table 20.2. Daemons

| Type | Name |
|---|---|
| Modeler | **zenmodeler** |
| Performance Collector | **zencommand** |