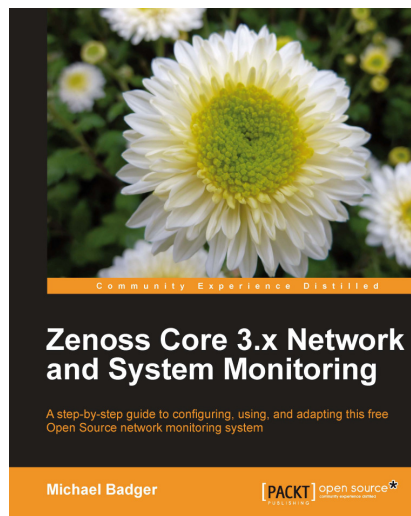


Zenoss Core 3.x Network and System Monitoring

Michael Badger



Chapter No. 3

"Device Setup and Administration"

In this package, you will find:

A Biography of the author of the book

A preview chapter from the book, Chapter NO.3 "Device Setup and Administration"

A synopsis of the book's content

Information on where to buy this book

About the Author

Michael Badger is a freelance technical communicator with a knack for helping other people understand and use their computer software and technology. In addition to writing a previous book about Zenoss Core: *Zenoss Core Network and System Monitoring*, Badger authored *Scratch 1.4: Beginner's Guide*, a Scratch programming tutorial.

He lives in north central Pennsylvania (United States) on a small farm and has recently taken to raising pastured chickens, honeybees, and pigs. Michael is searching for a way to integrate Zenoss Core into the hen house so that he can receive an alert each time an egg is laid.

For more information, visit www.badgerfiles.com/zenoss3.

There are so many people to thank, starting with my family. They tolerate my late nights and weekend work.

My team at Packt deserves a nod for finally helping me get this revision done. Thanks for the help Rakesh Shejwal and Jovita Pinto.

Then there are the reviewers. It's not easy to provide substantive critique of another person's work because it takes time and thoughtful consideration for you to want to make my work better. You should know that even though I did not incorporate all your suggestions, I considered them carefully.

For More Information:

www.packtpub.com/zenoss-core-3-x-network-and-system-monitoring/book

Zenoss Core 3.x Network and System Monitoring

For system administrators, network engineers, and security analysts, it is essential to keep a track of network traffic.

Zenoss Core is an enterprise-level systems and network monitoring solution that can be as complex as you need it to be. While just about anyone can install it, turn it on, and monitor "something", Zenoss Core has a complicated interface packed with features. The interface has been drastically improved over version 2, but it's still not the type of software you can use intuitively—in other words, a bit of guidance is in order.

The role of this book is to serve as your Zenoss Core tour guide and save you hours, days, maybe weeks of time.

This book will show you how to work with Zenoss and effectively adapt Zenoss for System and Network monitoring. Starting with the Zenoss basics, it requires no existing knowledge of systems management, and whether or not you can recite MIB trees and OIDs from memory is irrelevant. Advanced users will be able to identify ways in which they can customize the system to do more, while less advanced users will appreciate the ease of use Zenoss provides. The book contains step-by-step examples to demonstrate Zenoss Core's capabilities. The best approach to using this book is to sit down with Zenoss and apply the examples found in these pages to your system.

The book covers the monitoring basics: adding devices, monitoring for availability and performance, processing events, and reviewing reports. It also dives into more advanced customizations, such as custom device reports, external event handling (for example, syslog server, zensendevent, and Windows Event Logs), custom monitoring templates using SNMP data sources, along with Nagios, and Cacti plugins. An example of a Nagios-style plugin is included and the book shows you where to get an example of a Cacti-compatible plugin for use as a command data source in monitoring templates.

In Zenoss Core, ZenPacks are modules that add monitoring functionality. Using the Nagios plugin example, you will learn how to create, package, and distribute a ZenPack. You also learn how to explore Zenoss Core's data model using zendmd so that you can more effectively write event transformations and custom device reports.

Implement Zenoss Core and fit it into your security management environment using this easy-to-understand tutorial guide.

For More Information:

www.packtpub.com/zenoss-core-3-x-network-and-system-monitoring/book

What This Book Covers

Chapter 1, Network and System Monitoring with Zenoss Core, provides an overview of Zenoss Core's monitoring capabilities and system architecture.

In *Chapter 2, Discovering Devices*, we prepare our monitoring environment by configuring SNMP, WMI, SSH, and firewall ports. We'll add devices to Zenoss Core via the setup wizard, zenbatchload, and zendisc.

Chapter 3, Device Setup and Administration, configures devices so that we ensure we collect the proper monitoring information by organizing, configuring, and troubleshooting the monitoring properties.

Chapter 4, Monitor Status and Performance, monitors and graphs the performance of device components such as routes, windows services, IP services, processes, file systems, and network interfaces.

Chapter 5, Custom Monitoring Templates, explores custom monitoring templates by configuring various data sources, including SNMP, Nagios plugins, and Cacti plugins.

Chapter 6, Core Event Management, introduces us to processing events via the Event Console. We create custom event commands, learn how to create test events, and perform event mapping.

Chapter 7, Collecting Events, allows Zenoss Core to receive and process events from third-party sources, such as syslog, Windows Event Log, e-mail, and home-grown system administration scripts.

Chapter 8, Settings and Administration, covers common Zenoss Core administration tasks, such as managing users, the monitoring dashboard, backups, and updates.

Chapter 9, Extending Zenoss Core with ZenPacks, installs, creates, and packages add-on modules. ZenPacks extend the functionality of Zenoss Core.

Chapter 10, Reviewing Built-in Reports, reviews each of Zenoss Core's included reports to help us troubleshoot, analyze, and view our monitoring performance over time. It also creates custom graph and multi-graph reports.

Chapter 11, Writing Custom Device Reports, provides an in-depth look at Zenoss Core's custom device report functionality, including the use of zendmd to explore the Zenoss data model.

Appendix A, Event Attributes, lists the available event attributes in Zenoss Core.

Appendix B, Device Attribute, lists the attributes that we may use when working with our devices.

Appendix C, Example snmpd.conf, lists a sample snmpd.conf file.

For More Information:

www.packtpub.com/zenoss-core-3-x-network-and-system-monitoring/book

3

Device Setup and Administration

Based on our work in *Chapter 2, Discover Devices* Zenoss Core is now monitoring all the devices we added to our inventory. However, Zenoss Core is probably not monitoring the way we need it to, and if we look closely, we may notice that some devices have events associated with them. In this chapter, we're going to configure our devices so that we ensure we collect the proper information.

As we work through the chapter, we will:

1. Organize devices by class, location, system, and group
2. Model devices via SNMP, WMI, and Zenoss Plugins
3. Troubleshoot our Zenoss Core data collection
4. Administer device properties

We'll spend the bulk of this chapter fine-tuning our device inventory and getting it into monitoring shape. One of Zenoss Core's critical concepts is inheritance, which means that the devices inherit monitoring properties from their parent device class.

We can change the monitoring rules at the device level by tweaking individual device properties, called *zProperties*.

Let's start by taking a look at some of Zenoss Core's organizers.

For More Information:

www.packtpub.com/zenoss-core-3-x-network-and-system-monitoring/book

Organizing devices in Zenoss Core

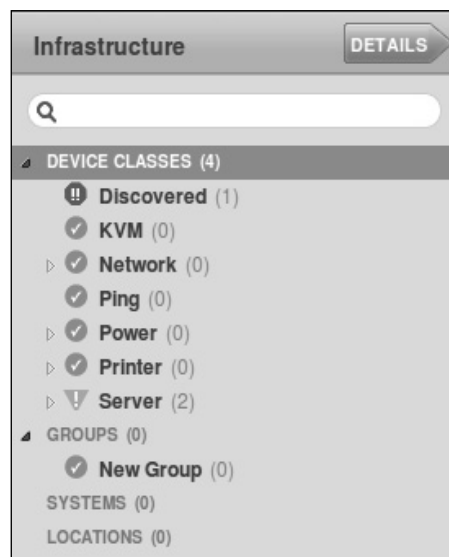
Zenoss Core provides multiple ways to organize our devices. We're going to take a closer look at Locations, Systems, Groups, Networks, and Classes. Collectively, these organizers allow us to describe devices in a way that's meaningful to your organization.

We can define the organizers to be as specific as we need them to be, and not all organizers are required. In other words, no one will care if you do not define any groups. The information that we do specify can be used to establish monitoring rules, event handling, and alerting rules.

Let's start by adding a location.

Locations

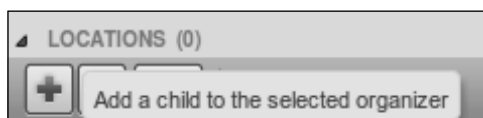
Zenoss Core does not include any location organizers by default. To add a location, click on the **Infrastructure** menu. A list of organizers displays in the left sidebar of the page, as seen in the following screenshot:



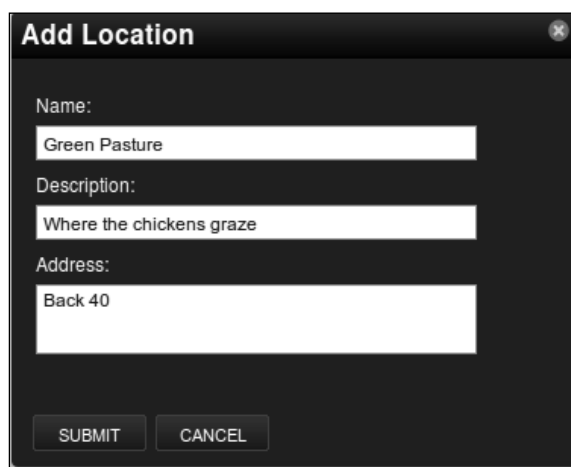
Potential location names are floors, buildings, wings, room numbers, or office locations. Use something that's meaningful to you.

To enter a new location:

1. Select **Locations** from the sidebar of organizers.
2. Click the plus sign (+) at the bottom of the sidebar to **Add a child to the selected organizer** to display the **Add Location** dialog:



3. Enter the **Name** of the location, a **Description**, and an **Address**.
4. Click on **SUBMIT** to add the location:

A screenshot of a dialog box titled 'Add Location'. It contains three text input fields: 'Name:' with the value 'Green Pasture', 'Description:' with the value 'Where the chickens graze', and 'Address:' with the value 'Back 40'. At the bottom are two buttons: 'SUBMIT' and 'CANCEL'.

Enter anything you think may be important about the location of that device. For example, phone extension, driving directions, or site contacts may be practical items to include in the description.

The real-world address may be something you want to record about a location, especially if you have multiple locations. Zenoss Core provides a separate **Address** field for each location name that ties into the Google Maps portlet.

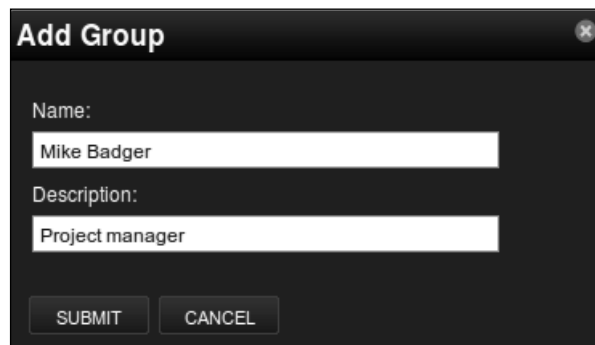
The Google Maps portlet displays your locations on a map and connects them for some nice dashboard eye candy. If a marketer happens to come by, be sure to show them the Google Maps portlet. You may make a friend. We'll cover the maps portlet in *Chapter 9, Extending Zenoss Core with ZenPacks*.

We can add as many locations as we need, or we can define a hierarchy of locations by adding a sub-location. To add a sub-location, click on the name of the location from the list and click on the **Add a child to the selected organizer** button.

Systems and Groups

Groups are closely related to Systems and how you use them is an exercise of your imagination or operational needs. As an example, you might use groups to define nodes on the org chart while the Systems describe the devices in terms of function. In my monitoring environment, I group websites by project manager, so my groups are names of people. I don't define any "systems" organizers, but you can. There's no wrong answer.

Systems and Groups are displayed in the same spot as the locations, by clicking on the **Infrastructure** menu. Adding either a system or a group works the same way, too. Click on either **System** or **Group**, and then click the **Add a child to the selected organizer** button. Enter the **Name** and **Description**, as shown in the following screenshot:

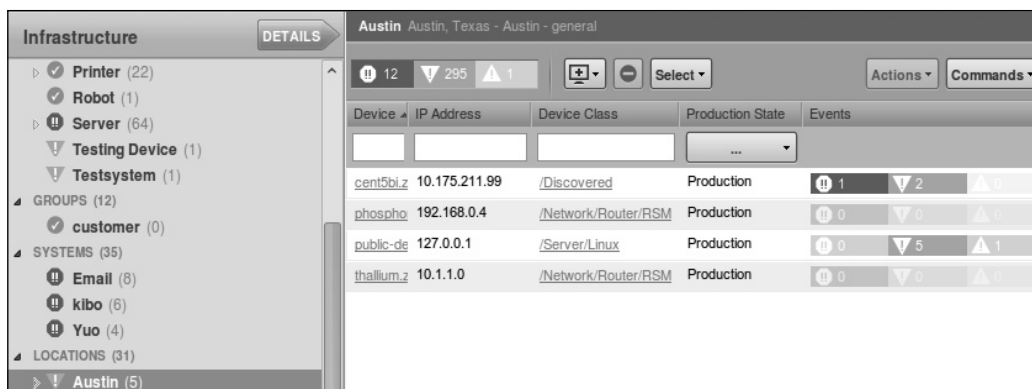


Adding a system is the same as adding a group, except that you select the **Systems** organizer in the sidebar first.

Organizer details

Before we look at classes, let's take a quick look at how we can work with Locations, Groups, and Systems in the Zenoss Core interface.

You will notice that as you click on an organizer, the device list filters to include all the devices assigned to an organizer. We haven't assigned any devices to an organizer yet, so the device list is empty. However the following screenshot shows an established Zenoss Core installation:



If you click on the **Details** button, the sidebar changes to display the following items: **Devices**, **Events**, **Administration**, **Map**, and **Modifications**.



As we move through the book, we will cover each of these items in more detail, so I'll only provide a brief overview now. If you click on the **Events** link, you see all the events associated with that organizer. We cover events in *Chapter 6, Core Event Management*.

Click on the **Administration** link to display a list of commands, maintenance windows, and device administrators. We cover administration in *Chapter 8, Settings and Administration*.

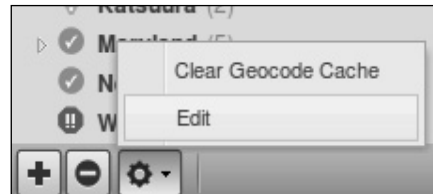
The **Map** page displays a map of the location based on the address information you entered for the organizer. The map requires a Google Maps key and we'll set that up in *Chapter 8, Settings and Administration*.

To see a list of changes for the device, click on the **Modifications** link.

To exit the **Details** page, click on the **See All** button.

Editing organizers

You can change the name or description of an organizer by selecting the organizer and then clicking **Edit**. You can find the Edit option by clicking on the Actions menu button (which looks like a sprocket) at the bottom of the **Infrastructure** sidebar. See the following screenshot:



Moving organizers

You can nest organizers by dragging and dropping one organizer into another one to create a hierarchy. You can only nest similar organizers, which means that you cannot move a system into a group or a group into a location.

Classes

Classes provide one of the most important organizers in Zenoss Core because we can use the classes to establish monitoring properties for groups of devices. Each device inherits the properties of its parent class. Using classes, we can define specific monitoring rules and settings via zProperties. All devices in a class share the same zProperties.

Classes also define what types of information Zenoss Core collects about a device by assigning a set of collector plugins. Like the zProperties, each class contains a set of collector plugins that can change from one class to another.



The Zenoss Plugins that we install on remote Unix servers are not the same as the collector plugins we configure for our devices.

Several classes exist to organize devices, events, services, and processes based on common groupings, but we'll stick to talking about devices right now.

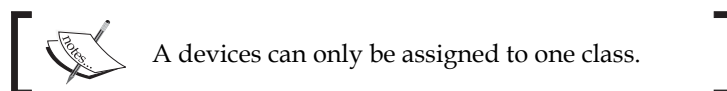
Viewing a list of device classes

You have probably already noticed that the list of device classes is available when you click on the **Infrastructure** menu. Zenoss Core includes a default hierarchy of classes; for now, we'll work within those default classes.

Next to each class is a number within parenthesis that indicates the number of devices that are associated with the class, including the sub-classes. The icon to the left shows the highest severity event for each class.



Click on the class name to display all the devices assigned to that class.



A hierarchical list of the default device classes in Zenoss Core follows. However you can add as many classes as you want.

List of device classes

- Discovered
- KVM
- Network
 - Router
 - Cisco
 - Firewall
 - RSM
 - Terminal Server
 - Switch
- Ping

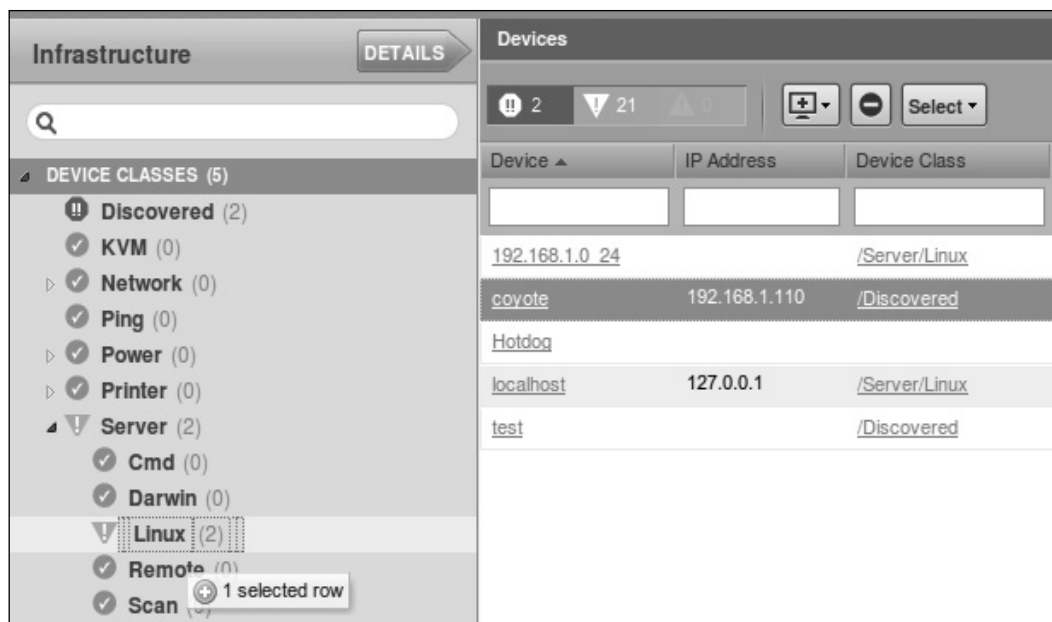
- Power
 - UPS
 - APC
- Printer
 - InkJet
 - Laser
- Server
 - Cmd
 - Darwin
 - Linux
 - Remote
 - Scan
 - Solaris
 - Windows

Assigning devices to a class

Depending on how you added devices, you may need to reclassify them. For example, if you auto discovered your network, all the devices in your inventory will be classified as **/Discovered**. There's no long term value in that.

To assign devices to a class:

1. Click on the **Infrastructure** menu to display a list of all devices.
2. Select one or more devices.
3. Drag and drop the selected devices onto the appropriate device class.
4. Click **OK** when prompted to confirm the move.



The screenshot shows the device named **coyote** being moved to the **/Server/Linux** class. Notice that the interface is giving you some feedback. You can only assign a device to one class, so the selected class is highlighted.

The tool tip box in the screenshot is showing **1 selected row**, which corresponds to the number of devices you're adding to the class. In the screenshot, only one device is being moved to **/Server/Linux**.

After you classify the devices, it's time to model them.

Modeling devices

Let's get our devices in shape to be modeled.

When we talk about Zenoss Core, two related terms often come up, monitoring and modeling. Monitoring refers to the availability of the device and answers the question, "Is the device accessible?". Modeling defines relationships between devices and identifies the components available on a device, such as services, interfaces, and file systems. It then collects varying amounts of data about each component.

Zenoss models devices via WMI, SNMP, SSH/Telnet, and port scan. Each class has a default set of collector plugins that tells Zenoss how to model the devices assigned to the class.

Remember in *Chapter 2, Discovering Devices* when we set up our servers to be monitored by setting up WMI, SNMP, and the Zenoss plugins? Now is the time to put that work to use. Feel free to go back and review that section now.

As we model our servers, routers, and other network devices, we'll be interested in setting the collector plugins and zProperties for each device.

Modeler plugins gather device information

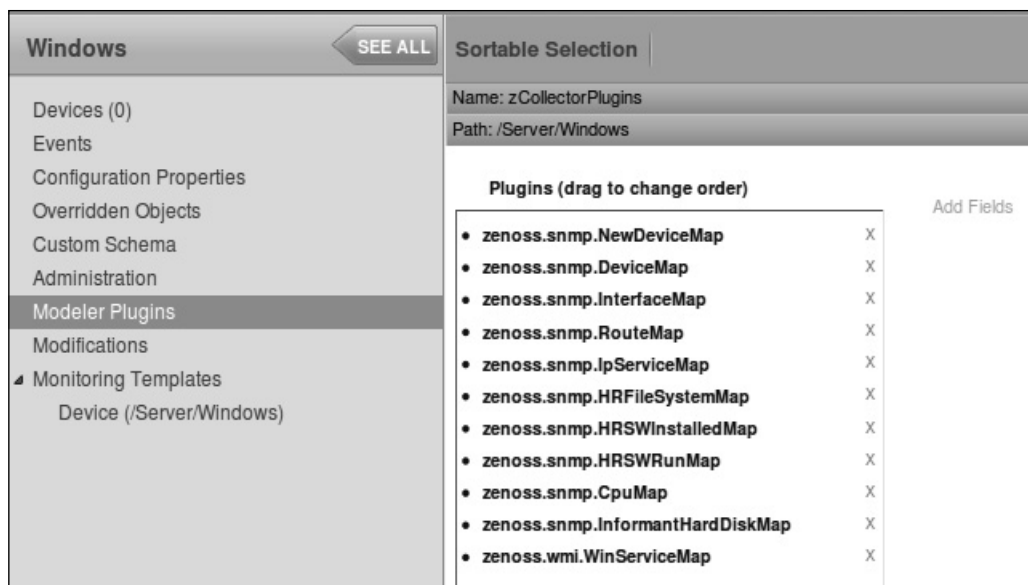
Each modeler plugin queries the device for a specific set of information that it will store in Zenoss Core's device database. What we collect depends on multiple factors, such as the capabilities of the device and the modeler plugins assigned to the device. In general, we can expect to collect information about the operating system, file systems, interfaces, routes, processes, IP services, CPU, and memory.



Modeler plugins can be assigned at the class level. However, we can configure exceptions from the class properties at the device level.

Let's examine these modeler plugins I've been talking about.

1. From the **Infrastructure** menu, select **Devices**.
2. Navigate to the **/Servers/Windows** class.
3. Click on the **Details** button to display a list of settings.
4. Find and click the **Modeler Plugins** link to display a sortable list of plugins.



A page showing the assigned collector plugins displays in the left column of the page with an **Add Fields** link on the right.

To see a list of unassigned plugins click on the **Add Fields** link.

The plugin names are intuitive in that the name suggests the type of information we expect to collect. For example, `zenoss.snmp.IpServiceMap` uses SNMP to return a list of active IP services on the device, such as HTTP. The Dell-specific plugins retrieve more detailed information from Dell devices using OpenManage, and the HP plugins provide more information about devices using Insight Management agents.

The plugins follow a three part naming convention. The first part identifies the author (for example, `zenoss`). The second part lists the collection method (for example, `SNMP`, `WMI`, `cmd`). The final part identifies the specific information collected by the plugin (for example, `IpServiceMap`, `CpuMap`, `uname`).

Actually, the command plugins, identified by the "cmd" in the name, have an optional field to specify system architecture. For example, `cmd.linux` applies to Linux servers while `cmd.darwin` plugins apply to OSX.

To remove a plugin from the assigned plugin list, click on the **x** next to the plugin name. To assign a plugin, drag the plugin name from the available list to the assigned list.



Any change we make to the collector plugins at the class level will be applied to all devices in the class. If we have one-off changes, we should make the changes at the device level.

Assigning modeler plugins

Take the time now to review the devices in your inventory and set the appropriate collector plugins. If you're not sure where to start, consider the following questions as a starting point:

- Will you monitor with WMI? If not, remove the `zenoss.wmi.WinServiceMap` plugin.
- Does the device support SNMP? If not, remove the `snmp` plugins.
- Is the server a Dell or HP? If so, add the `PowerwareDeviceMap` or `SysedgeDiskMap` and `SysedgeMap` plugins.
- Are you monitoring a remote OSX server with the Zenoss Plugins? If so, add the `zenoss.cmd.darwin` plugin.
- Plan to rely on port scan? Then use the port scan plugin.

Obviously, we can't cover every possible scenario; however that should get you started.

Troubleshooting data collection

Wouldn't it be nice if everything just worked? Unfortunately, we can expect to make a mistake or two.

Troubleshooting SNMP problems

If you see an event in Zenoss that reports SNMP agent down, we first need to find out whether the issue is with Zenoss Core or with the device.



Not all network devices support SNMP. So, before you beat yourself up troubleshooting, make sure your device supports SNMP.

Running snmpwalk

Zenoss Core includes a command that will run the `snmpwalk` command on the selected device. The `snmpwalk` command will query the device using the SNMP community string we assigned to the device and will try to get a list of the OIDs. Let's try it.

Navigate to **Infrastructure** and select **Devices**. Select a device (by clicking on the device name) that should respond to SNMP requests. The device's **Overview** page displays.

1. At the bottom of the sidebar, expand the **Commands** menu and select **snmpwalk**.
2. The command runs against the device and returns the result, as shown in the following screenshot:

```

snmpwalk
===== coyote =====
snmpwalk -v1 -cpublic 192.168.1.110 system
SNMPv2-MIB::sysDescr.0 = STRING: Linux coyote 2.6.20-15-server #2 SMP Sun Apr 15 07:41:34 UTC 2007 i686
SNMPv2-MIB::sysObjectID.0 = OID: NET-SNMP-MIB::netSnmpAgentOIDs.10
DISMAN-EVENT-MIB::sysUpTimeInstance = Timeticks: (141975125) 16 days, 10:22:31.25
SNMPv2-MIB::sysContact.0 = STRING: Root (configure /etc/snmp/snmp.local.conf)
SNMPv2-MIB::sysName.0 = STRING: coyote
SNMPv2-MIB::sysLocation.0 = STRING: Unknown (edit /etc/snmp/snmpd.conf)
SNMPv2-MIB::sysORLastChange.0 = Timeticks: (18) 0:00:00.18
SNMPv2-MIB::sysORID.1 = OID: IF-MIB::ifMIB
SNMPv2-MIB::sysORID.2 = OID: SNMPv2-MIB::snmpMIB
SNMPv2-MIB::sysORID.3 = OID: TCP-MIB::tcpMIB
  
```

The previous screenshot indicates that Zenoss Core was able to retrieve data from the device successfully. If you have problems, the `snmpwalk` command will either timeout or be refused.

If you're having problems connecting, check the SNMP Community attribute (`zSnmpCommunity`) from the Configuration Settings of the device to make sure it's correct.

If problems continue, let's move to the device itself and run the `snmpwalk` command, assuming we're working with a Linux server. The `snmpwalk` command is included in the SNMP utilities we installed in *Chapter 2, Discovering Devices*.

Run this command on the problem device:

```
snmpwalk -c public -v 2c localhost
```

If the command is successful, pages of results will scroll by the terminal. You should check to make sure the device is allowing incoming connections on port 161/UDP and that both the device and Zenoss Core are using the correct community strings.

Is the SNMP daemon running on Linux servers?

You should also check to make sure the `snmpd` daemon is running. Run the following command to check:

```
ps aux | grep snmpd
```

We expect to see `/usr/bin/snmpd` running in the process list. If you don't see `snmpd`, then install it as per the *Prepare devices for monitoring* section in *Chapter 2, Discovering Devices*.

If you are still having problems, review the SNMP zProperties for the device. You can try different SNMP versions and make sure SNMP monitoring is enabled. To find the zProperties for a device, navigate to the device's status page. Then click on the **Configuration Properties** link.

A description of the zProperties of all devices is included at the end of this chapter.

SNMP problems on Windows

Assuming the `snmpwalk` fails when run from Zenoss Core, make sure SNMP is installed on the server (see *Chapter 2, Discovering Devices*).

The built-in Windows SNMP client doesn't provide as much information as Net-SNMP does for our Unix-based systems. You may be wondering why Zenoss Core isn't collecting file system information or CPU statistics. You need a third-party SNMP agent for Windows. Try the free SNMP agent from <http://www.snmp-informant.com>.

Troubleshooting WMI problems

The first thing you should do is to make sure you have WMI installed as per the instructions in *Chapter 2, Discovering Devices*, and have verified the setup.

Zeneventlog—unable to connect to Windows

If you see an event that indicates zeneventlog is unable to connect to Windows, that's an indication Zenoss Core is not able to authenticate to the Windows server. Check the Configuration Properties (zProperties) of the device to ensure you have the correct username and password set.

The following screenshot shows the WMI specific zProperties:

zWinEventlog	False ▾	boolean /
zWinEventlogMinSeverity	2	int /
zWinPassword	*****	password /
zWinUser	administrator	string /
zWmiMonitorIgnore	True ▾	boolean /
zXmlRpcMonitorIgnore	True ▾	boolean /

Remember, the zWinUser and zWinPassword properties must be an administrator on the Windows server you're monitoring. It's possible that each of your Windows servers could require a different username and password.

The zWinUser property can be either a local or a domain account. If you use a domain account, specify the domain for the zWinuser value (for example, mojoactive\administrator).

Zenoss Core does not collect WMI data

It's possible that Zenoss Core isn't collecting any data from the Windows Servers via WMI, but there are no events reported. You can check the following zProperties at the device or class level:

- Make sure the device is using the `zenoss.wmi.WinServiceMap` collector plugin
- Set the **zWmiMonitorIgnore** property to **False** in the **Configuration Properties** for the device or device class
- If you want to collect event logs, set the **zWinEventlog** property to **True** in the **Configuration Properties** for the device or device class

Troubleshooting Zenoss Plugins

In *Chapter 2, Discovering Devices* we installed the Zenoss plugins on Unix hosts that we may not be able to monitor with SNMP, such as remote Linux or OSX servers. If you're encountering problems, there are several `zProperties` that affect how successfully you can monitor remote machines. See the following screenshot:

zCommandCommandTimeout	15.0	float	/
zCommandCycleTime	60	int	/
zCommandExistenceTest	test -f %s	string	/
zCommandLoginTimeout	10.0	float	/
zCommandLoginTries	1	int	/
zCommandPassword		password	/
zCommandPath	/usr/local/zenoss/libexec	string	/
zCommandPort	22	int	/
zCommandProtocol	ssh	string	/
zCommandSearchPath		lines	/
zCommandUsername		string	/

There are several points of failure, but the most probable error is that the **zCommandPath** value is not valid if you installed the Zenoss Plugins as I described. The `zenplugin.py` file is installed to `/usr/bin`. And as you can see, the **zCommandPath** is pointing to `/usr/local/zenoss/libexec`. This means that each time Zenoss Core tries to collect data from the device using the remote command, it's trying to run `/usr/local/zenoss/libexec/zenplugin.py` on the remote server.

Change the **zCommandPath** property to reflect the actual path to `zenplugin.py` on the remote server, such as `/usr/bin`.

In order for Zenoss Core to connect to the remote server, it must supply valid user credentials. Check the **zCommandUsername** and the **zCommandPassword** properties to ensure they contain a valid username and password for the remote server.

A class of its own

Zenoss Core provides the `/Server/Cmd` class specifically for monitoring remote machines. The collector plugins are specifically designed to interact with `zenplugin.py` and retrieve data about the remote server. The following screenshot shows the collector plugins assigned to the `/Server/Cmd` class.

Sortable Selection

Name: zCollectorPlugins

Path: /Server/Cmd

Plugins (drag to change order)

• zenoss.cmd.uname	X
• zenoss.cmd.df	X
• zenoss.cmd.linux.ifconfig	X
• zenoss.cmd.linux.memory	X
• zenoss.cmd.linux.netstat_an	X
• zenoss.cmd.linux.netstat_rn	X
• zenoss.cmd.linux.process	X
• zenoss.cmd.darwin.cpu	X
• zenoss.cmd.darwin.ifconfig	X
• zenoss.cmd.darwin.memory	X
• zenoss.cmd.darwin.netstat_an	X
• zenoss.cmd.darwin.process	X
• zenoss.cmd.darwin.swap	X

Save Delete

You won't find any SNMP collectors here.

Device administration

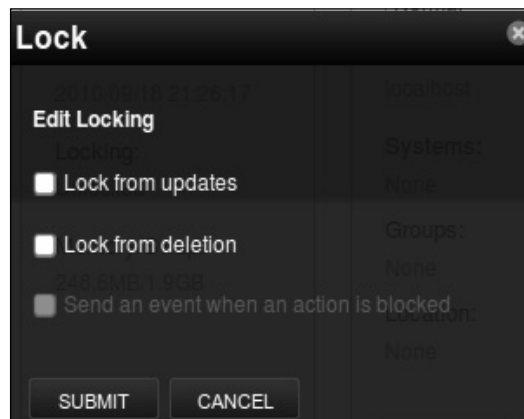
In this section, we'll take a look at some basic device administration tasks, including: rename a device, delete a device, reset the IP address, and lock the device's configuration.

Locking or unlocking a device

Zenoss Core automatically polls the devices in our inventory and remodels the devices when it finds changes. To prevent changes to the device properties, we can lock the configuration, and we can also lock the device from being deleted from the inventory.

To change the lock status of a device:

1. From the **Device Overview** page, select **Lock** from the **Actions** menu.
2. Select from these choices, as shown in the following screenshot:
 - **Lock from updates**
 - **Lock from deletion**
 - **Send event when actions are blocked**



3. The device status page displays after we choose a locking option.

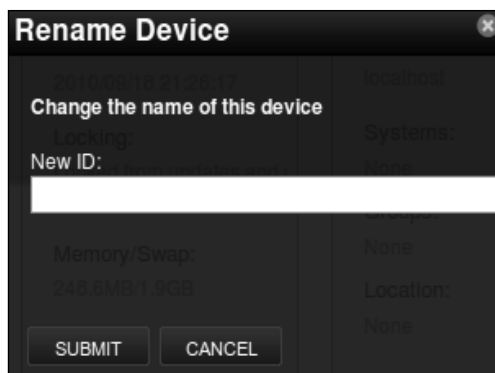
The options should be self explanatory. If you choose to send an event when actions are blocked by a lock, the event will show up in the event console for the device.

Renaming a device

Zenoss automatically detects and populates the device name, but we can name the device anything we want. On my test network, I prefer to use the names of furbearers.

1. From the **Device Overview** page, select **Rename Device** from the **Actions** menu.
2. Enter the new name (for example, Coyote) in the **ID** field of the **Rename Device** dialog.
3. Click on **OK** to save the change.

On the **Device Status** page, the device information updates to reflect the new name. Even the breadcrumb navigation changes to reflect the new name.



Now, you can manage your devices using whatever slang you wish.

Resetting the IP address

Sometimes we need to move machines around and allocate new IP addresses. At other times, we may try to monitor a server only to discover that it has a dynamic IP address. Since Zenoss Core requires a static IP address on the monitored device, we need to assign an IP address to the server, and therefore, IP information will need to be updated in Zenoss Core.

To change the IP address:

1. From the **Device Overview** page, select **Reset/Change IP** from the **Actions** menu.
2. Enter the new resolvable hostname or IP address in the **IP Address** field of the **Reset/Change IP** dialog box (shown in the following screenshot) or leave it blank to allow Zenoss to lookup the IP based on the device name.
3. Click **OK** to save the change.



Push changes

After we make changes to the device, we can "push" the changes to the collectors right away instead of waiting for Zenoss Core to remodel the device. From the **Device Overview** page, select **Push Changes** from the **Actions** menu.

Zenoss Core confirms the action with a status message in the bottom-right corner of the page.

The collectors in Zenoss Core define settings that determine how each device assigned to a collector is monitored. For example, we can configure the default cycle times for the modeler protocol (SNMP, WMI, Ping). On Zenoss Core, the default collector is localhost. Zenoss Core supports only one collector on the same host.

Deleting devices

Given enough time, we will find devices in our inventory that we want to delete. For example, maybe we accidentally added a bunch of workstations with dynamic IP addresses that go up and down. There are many other reasons to delete a device from Zenoss Core, but I trust you have a firm grasp on when deleting a device from the inventory is necessary.



If you think you might want to monitor the device in the future, you should consider setting the production state to decommissioned. This keeps all the historical data and retains the device in the inventory; however, it's no longer monitored.

To remove a device from inventory:

1. Click on the **Infrastructure** menu and select the devices you want to remove.
2. From the **Device Overview** page, select **Delete** from the **Actions** menu.
3. Select the **Remove Devices** button (it looks like a minus sign) located at the top of the device list.
4. The **Remove Devices** dialog displays and prompts you to select from the following options:
 - Delete current events for these devices
 - Delete performance data for these devices
5. Make the appropriate selections and click on **OK** to confirm the delete.
6. The device will no longer show in inventory and Zenoss Core will no longer monitor it.

If you choose to retain the performance data by not deleting it, the data will be available to the device should we decide to add the device back to Zenoss Core at a later time. We'll review performance data in more detail in *Chapter 5, Custom Monitoring Templates* and *Chapter 6, Core Event Management*.

zProperties defined

Throughout this chapter, we've seen several specific examples of zProperties, but there are many properties that impact how we monitor. The following table lists the device zProperties along with a brief description of the property. The zProperties are accessible via the Configuration Properties link at the device or class level.

zProperty	Description
zCollectorClientTimeout	Set the collector timeout in seconds. The default value is 180 seconds.
zCollectorDecoding	Specify the character encoding. The default is latin-1.
zCollectorLogChanges	Set to true to log changes and false to not log changes to the collector.
zCollectorPlugins	Click the Edit link to open the collector plugin selection page.
zCommandTimeout	Time in seconds to wait for a command to finish. The default is 15.
zCommandCycleTime	Specifies a time in seconds to cycle through zCommands. The default is 60.
zCommandExistanceTest	Test to see if a command exists on the monitored device. The default is 'test -f %s'.
zCommandLoginTimeout	Wait the specified number of seconds for a login prompt to display. The default is 10.
zCommandLoginTries	Attempt to log in to the remote sever the specified number of times. The default is 1.
zCommandPassword	Enter the password for the user's shell account on the monitored device.
zCommandPath	When you specify a command, Zenoss Core searches the specified path for the command. You can enter fully-qualified commands here.
zCommandPort	The port the monitored system uses for remote connections. The default is 22 for SSH. Specify 23 for telnet.
zCommandProtocol	Specify the protocol to use (Telnet, SSH). The default is SSH.
zCommandSearchPath	Not currently used.

zProperty	Description
zCommandUsername	Enter the user name for the remote device.
zDeviceTemplates	Enter the templates by name to use in order to display information. The default is device.
zFileSystemMapIgnoreNames	Enter the names of the file systems to ignore. For example: /boot.
zFileSystemMapIgnoreTypes	Specify the file systems you want Zenoss Core to ignore.
zFileSystemSizeOffset	Adjust the file system capacity values reported by Net-SNMP. Useful when the Linux server and Net-SNMP report different capacities.
zHardDiskMapMatch	A regular expression to match the names of hard disks.
zIcon	Each device class has a default icon that can be changed as necessary.
zIfDescription	Displays the interface description on the Interfaces table of the OS tab. Select either true or false. The default is false.
zInterfaceMapIgnoreNames	Enter the names of the interfaces to ignore. For example: lo.
zInterfaceMapIgnoreTypes	Enter the type of interfaces to ignore. For example: local loopback.
zIpServiceMapMaxPort	Specify the maximum port number to port scan. The default is 1024.
zKeyPath	Specify the path to the user's public key file for use with public-key authentication.
zLinks	Enter HTML or TALEX expressions to display content on the device's status page. For example, you can create a link to a router's administration console that will display on the Device Status page.
zLocalInterfaceNames	A regular expression match to identify local interface names. The default expression looks for lo (loopback) and vmnet (VMware).
zLocalIpAddresses	A regular expression match to identify local IP address.
zMaxOIDPerRequest	Specify the number of OIDs Zenoss collects with a single query. The default is 40.
zNmapPortscanOptions	Specify the options to use for the zenoss . portscan . IpServiceMap modeler plugin.
zPingMonitorIgnore	Select true to not ping the device or false to ping the device.

zProperty	Description
<code>zProdStateThreshold</code>	Monitor a service that is higher than the production state listed. Possible values include 1000 (Production), 500 (Pre-Production), 400 (Test), 300 (Maintenance), and -1 (Decommissioned).
<code>zPythonClass</code>	Not currently used.
<code>zRouteMapCollectOnlyIndirect</code>	Set to true to collect only the indirect routes. Default is false.
<code>zRouteMapCollectOnlyLocal</code>	Set to true to collect only the local routes. Default is false.
<code>zRouteMapMaxRoutes</code>	Specify the maximum number of routes to map. The default is 500.
<code>zSnmpAuthPassword</code>	Specify SNMP password, if applicable.
<code>zSnmpAuthType</code>	If using <code>zSnmpAuthPassword</code> , select either MD5 or SHA authentication protocol.
<code>zSnmpCommunities</code>	List of communities Zenoss tries to collect information for. The defaults are public and private. Enter more as needed.
<code>zSnmpCommunity</code>	The default community name on the monitored device.
<code>zSnmpMonitorIgnore</code>	Set whether or not Zenoss should monitor the device with SNMP. Defaults to false.
<code>zSnmpPort</code>	The SNMP communication port. It defaults to port 161.
<code>zSnmpPrivPassword</code>	Enter the security user's password.
<code>zSnmpPrivType</code>	Select either AES or DES encryption.
<code>zSnmpSecurityName</code>	Enter the security user's name.
<code>zSnmpTimeout</code>	Length of time in seconds that Zenoss waits for a response from the remote SNMP agent. Defaults to 2.5.
<code>zSnmpTries</code>	Number of times Zenoss tries to connect via SNMP before reporting a failure.
<code>zSnmpVer</code>	The version of SNMP. Available options are 1, 2c, and 3. Defaults to 1.
<code>zSshConcurrentSessions</code>	Set the number of concurrent SSH sessions that Zenoss Core will make. The default is 10.
<code>zStatusConnectTimeout</code>	Specifies the time in seconds for an IP service to respond before the service is marked as down. The default is 15.
<code>zSysedgeDiskMapIgnoreNames</code>	Not currently used.

zProperty	Description
zTelnetEnable	On Cisco routers, send the enable command to enable command collection. Default is false.
zTelnetEnableRegex	Match the enable prompt with the specified regular expression.
zTelnetLoginRegex	Match the login prompt with the specified regular expression.
zTelnetPasswordRegex	Match the password prompt with the specified regular expression.
zTelnetPromptTimeout	Specify the time in seconds to wait for the login prompt to display.
zTelnetSuccessRegexList	Match the command prompt with the specified regular expression.
zTelnetTermLength	Select true to enable Telnet terminal length.
zWinEventLog	Specifies whether or not Zenoss collects the Windows event log. Default is false.
zWinEventLogMinSeverity	Collect all Windows event logs that match the specified severity. Enter a value between 1 and 5, where 1 is the most severe. The default is 2.
zWinPassword	Enter the Windows user's password.
zWinUser	Enter the username of an account on the monitored Windows system.
zWmiMonitorIgnore	Set to true to ignore WMI monitoring and set to false to monitor WMI services.
zXmlRpcMonitorIgnore	Set to true to enable XML/RPC monitoring.

Summary

At this point, the devices in your inventory should be organized and configured for monitoring. We defined the monitoring settings by assigning devices to classes, adding/removing collector plugins, and fine tuning the zProperties for the class or the individual device.

As we work with our devices and develop our Zenoss Core monitoring environment, we may realize that we want to change some of the work we did in this chapter. And that's no problem.

In the next chapter, we'll dig into the availability and performance data that Zenoss Core collects for each component.

Where to buy this book

You can buy Zenoss Core 3.x Network and System Monitoring from the Packt Publishing website: <http://www.packtpub.com/zenoss-core-3-x-network-and-system-monitoring/book>

Free shipping to the US, UK, Europe and selected Asian countries. For more information, please read our [shipping policy](#).

Alternatively, you can buy the book from Amazon, BN.com, Computer Manuals and most internet book retailers.



www.PacktPub.com

For More Information:

www.packtpub.com/zenoss-core-3-x-network-and-system-monitoring/book