



Zenoss Service Impact User Guide

Release 5.3.1

Zenoss, Inc.

www.zenoss.com

Zenoss Service Impact User Guide

Copyright © 2018 Zenoss, Inc. All rights reserved.

Zenoss, Own IT, and the Zenoss logo are trademarks or registered trademarks of Zenoss, Inc., in the United States and other countries. All other trademarks, logos, and service marks are the property of Zenoss or other third parties. Use of these marks is prohibited without the express written consent of Zenoss, Inc., or the third-party owner.

Amazon Web Services, AWS, and EC2 are trademarks of Amazon.com, Inc. or its affiliates in the United States and/or other countries.

Flash is a registered trademark of Adobe Systems Incorporated.

Oracle, the Oracle logo, Java, and MySQL are registered trademarks of the Oracle Corporation and/or its affiliates.

Linux is a registered trademark of Linus Torvalds.

RabbitMQ is a trademark of Pivotal Software, Inc.

SNMP Informant is a trademark of Garth K. Williams (Informant Systems, Inc.).

Sybase is a registered trademark of Sybase, Inc.

Tomcat is a trademark of the Apache Software Foundation.

VMware is a registered trademark or trademark of VMware, Inc. in the United States and/or other jurisdictions.

Windows is a registered trademark of Microsoft Corporation in the United States and other countries.

All other companies and products mentioned are trademarks and property of their respective owners.

Part Number: 1210.18.100.25

Zenoss, Inc.
11305 Four Points Drive
Bldg 1 - Suite 300
Austin, Texas 78726

Contents

About this guide.....	4
Chapter 1: Introduction to Service Impact.....	5
Terminology and concepts.....	5
Service events.....	7
Service models, nodes, and graphs.....	8
State propagation policies.....	13
Service model definition process.....	18
Chapter 2: Tutorial: Creating a service model.....	19
Define the service to model.....	20
Create logical node and subservice members for Internet connections.....	25
Create members for major network segments.....	27
Create summary members for critical paths.....	28
Create a member for the network service.....	28
Create a member for the service model.....	29
Send events to fake devices.....	29
Remove tutorial service model members.....	30
Chapter 3: Exporting service models from one system to another.....	32
Exporting a service model.....	34
Importing service model definitions.....	34
Chapter 4: Browser interface resources.....	47
Adding an Impact Services portlet.....	47
Service Impact home page.....	47
Members view.....	49
Add to Service dialog box.....	49
Impact Events.....	50
Impact View.....	51
Impact Policies dialog box.....	54
Logical node details view.....	57
Chapter 5: Configuring Service Impact.....	59
Production state propagation threshold.....	59
Service Impact server configuration files.....	60
Targeted graph update.....	62

About this guide

Zenoss Service Impact User Guide provides information about using and administrating Zenoss Service Impact (Service Impact).

Audience

This guide is intended for system administrators who have experience with Zenoss Resource Manager (Resource Manager), a working knowledge of Linux system administration, and an understanding of their data center environment.

Related publications

Title	Description
<i>Zenoss Service Impact Installation and Upgrade Guide for Resource Manager 5.x and 6.x</i>	Describes how to install Service Impact with a Resource Manager version 5 or 6 deployment.
<i>Zenoss Service Impact Installation and Upgrade Guide for Resource Manager 4.2</i>	Describes how to install Service Impact with a Resource Manager version 4.2 deployment.
<i>Zenoss Service Impact Release Notes</i>	Describes known issues, fixed issues, and late-breaking information not already provided in the published documentation set.

Additional information and comments

If you have technical questions about this product that are not answered in this guide, visit the [Zenoss Support](#) site.

Zenoss welcomes your comments and suggestions regarding our documentation. To share your comments, please send an email to docs@zenoss.com. In the email, include the document title and part number. The part number appears at the end of the list of trademarks, at the front of this guide.

1

Introduction to Service Impact

Think of a *service* as any business activity or interface that is provided to users to accomplish a goal. Examples include systems that represent specific functions like Payroll processing, a Customer Relationship Management (CRM) tool, departments in a company, such as Finance or Marketing, and a company website.

A service's availability and performance characteristics are provided by key infrastructure entities, such as databases and application servers, and the hierarchy of those key entities' dependencies on other devices and components, such as compute, storage, networking, and so on.

Most critical applications have infrastructure redundancies so that service availability continues if a subset of devices fail. When IT Operations staff are triaging IT infrastructure problems, it is critical to isolate the source and understand the effect on the service and the business. This effort is called the *mean-time-to-know* (MTTK). According to industry experts, MTTK is responsible for over 70% of the mean-time-to-repair problems (MTTR). For instance, if an application server crashes, the following questions must be answered to triage the issue:

- What is the priority? What services does that server support and are those services mission critical or low priority?
- Are enough redundant application servers available to carry the load, or is the service effectively offline?
- If multiple symptoms appear, which symptom is the root cause? This information is needed to assign the right team to start working on a resolution.

Service Impact enables you to quickly create *service models* of services and applications in your environment. Service Impact can display service models as a *service model impact dependency graph*.

To help you triage service problems, Service Impact performs near-real-time dependency tracking of the individual components of a service. Service Impact also maintains the availability and performance state information of your services. When new events occur at any point in a service, Service Impact knows the origins of the events and assesses whether to generate a *service event* based on the extent to which the service is "impacted." A service event differs from an infrastructure event.

Terminology and concepts

Service Impact and this document use the following terminology and concepts.

dynamic service

A Service Impact entity that is used to monitor the status of a service. A dynamic service has a name and attributes, and is defined by the 0 or more members (devices, components, component groups, logical nodes, organizing groups, or other dynamic services) that you add to its service model.

See also *service*.

dynamic service context

A dynamic service's context includes all nodes in the dynamic Service Impact graph and the set of that service's context-specific policy configurations that are applied to nodes. These configurations override each node's global policy. The service context includes policies and rules that are specific to a dynamic service and global policies and rules that govern the operational decisions and actions on that service. For example, the conditions under which events propagate to eventually generate service events. An entity such as a device or component can participate as a node in multiple service contexts. When propagating an entity event, the rules and policy decisions being made can differ depending on each service context. For example, device D participates in service context X and Y. An entity event on D might propagate to generate a service X service event but not propagate in service Y's context.

dynamic Service Impact graph

Visual display of all entities that directly or indirectly impact a service. It is the combination of all service model members and each of their service model member impact graphs. On the Resource Manager **SERVICES** page, the **Impact View** provides a visual representation of a Service Impact graph.

Each unique entity exists only once in a Service Impact graph, even if the entity affects other service model members. For example, a database and an application server might depend on the same file system. However, in each dynamic Service Impact graph, the file system exists only once.

dynamic service model

Sometimes shortened to *service model*, it defines the service model members (devices, components, component groups, logical nodes, organizing groups, and other dynamic services) that are manually associated with a dynamic service by using the **Member View** on the Resource Manager **SERVICES** page.

For each service model member, the set of installed ZenPack use domain knowledge to automatically identify dependencies for each member and recursively for each of the member's dependent entities. For example, a ZenPack that manages an application server identifies that it depends on the Linux operating system. The Linux ZenPack identifies that the OS depends on file systems and a virtual machine, and that those entities depend on other entities an application server depends on the operating system, file systems, and virtual machine, and those members depend on other members, and so on until the entire impact graph for the service model member is modeled.

The set of impact graphs for all service model members is combined to generate the dynamic Service Impact graph. The Service Impact graph is displayed on the **Impact View** of the Resource Manager **SERVICES** page.

The service model and the generated Service Impact graph are different but related. You can manage the set of members in the service model. However, the dependencies and the Service Impact graph are managed by the ZenPacks that are specific to each entity.

dynamic service model member

Any Resource Manager entity that is part of a dynamic Service Impact graph. An entity can be a device, component, component group, logical node, organizing group, or another dynamic service.

See also *node*.

entity event

Any event that is related to a Resource Manager entity and displayed on the **EVENTS** page.

entity impact graph

The set of infrastructure entities that impact another entity's performance and availability. For each infrastructure entity, the graph includes their impact dependency entities and so on. Resource Manager internal processing automatically generates and maintains entity impact graphs. You cannot manually define or adjust impact graphs. However, you can manage the impact policy gateway rules for any node both globally and in individual dynamic service contexts.

Sometimes shortened to *impact graph*.

impact chain

See *service event impact chain*.

node

Any entity and its position, dependencies, and entities that depend on it within a Service Impact graph, including the dynamic service context itself, service model members, and member impact graph infrastructure entities.

root cause analysis (RCA)

See *service event root cause analysis*.

service

Any business activity or interface that is provided to users to accomplish a goal. Examples include systems that represent specific functions like payroll processing, a customer relationship management (CRM) tool, departments in a company, such as Finance or Marketing, and a company website. These logical entities can be represented as dynamic services by Service Impact in Resource Manager.

service event

A Service Impact-created event against a dynamic service. They are created in reaction to the propagation of one or more availability or performance entity events through nodes in the Service Impact graph. Service events contain a list of all impact chains and the relative probability of being the root cause.

service event impact chain

The propagation path from the entity that had an event through the Service Impact graph, applying node impact propagation policy gateways, until it reaches the dynamic service context node. The impact chain is generated for each originating entity event and displayed in the details of a service event.

service event root cause analysis

Displayed in the details for each service event to show the list of originating entity events and their impact chains that resulted in an impact to the dynamic service. Each event has an associated confidence value to help you identify the most likely root cause of the service problem.

service model

See *dynamic service model*.

Service Impact graph

The impact graph for an individual service model member.

targeted graph update

When Service Impact receives information about a device that is needed for an event, it targets modeling of just the device's subgraphs. To allow the modeling workflow to complete, a "placeholder" is immediately created. This process enables Service Impact graphs to be quickly refreshed without waiting for remodeling of all Service Impact graphs.

tile

The visual representation of any entity or node that participates in a Service Impact graph. The tile presents a node's status and other details and actions.

Service events

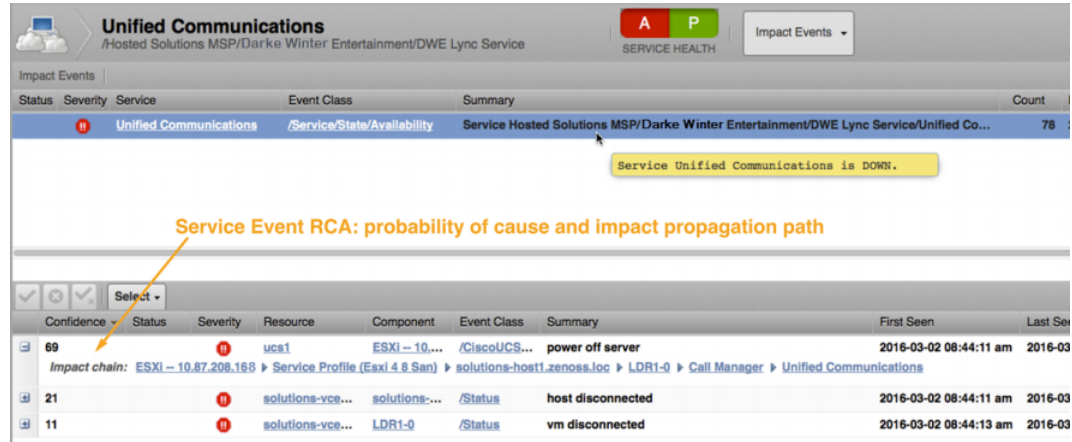
A service event provides the list of entity events and the daisy chain of how dependencies on that entity affected other entities and eventually the service itself.

Service Impact performs root-cause analysis (RCA) and ranks each source entity event by its probability of being the root cause of the service event. In a service event, Service Impact identifies the affected service and the extent of the impact, the infrastructure dependencies and sub-dependencies that affect the service, and the most likely root cause. Service Impact often enables you to triage and diagnose problems before end-user

complaints occur. You can then provide end users with confidence that you are aware of the issue and causes and are working a resolution.

The following figure shows a service event and the RCA. The **Confidence** column displays the probability ranking.

Figure 1: Service event and root cause

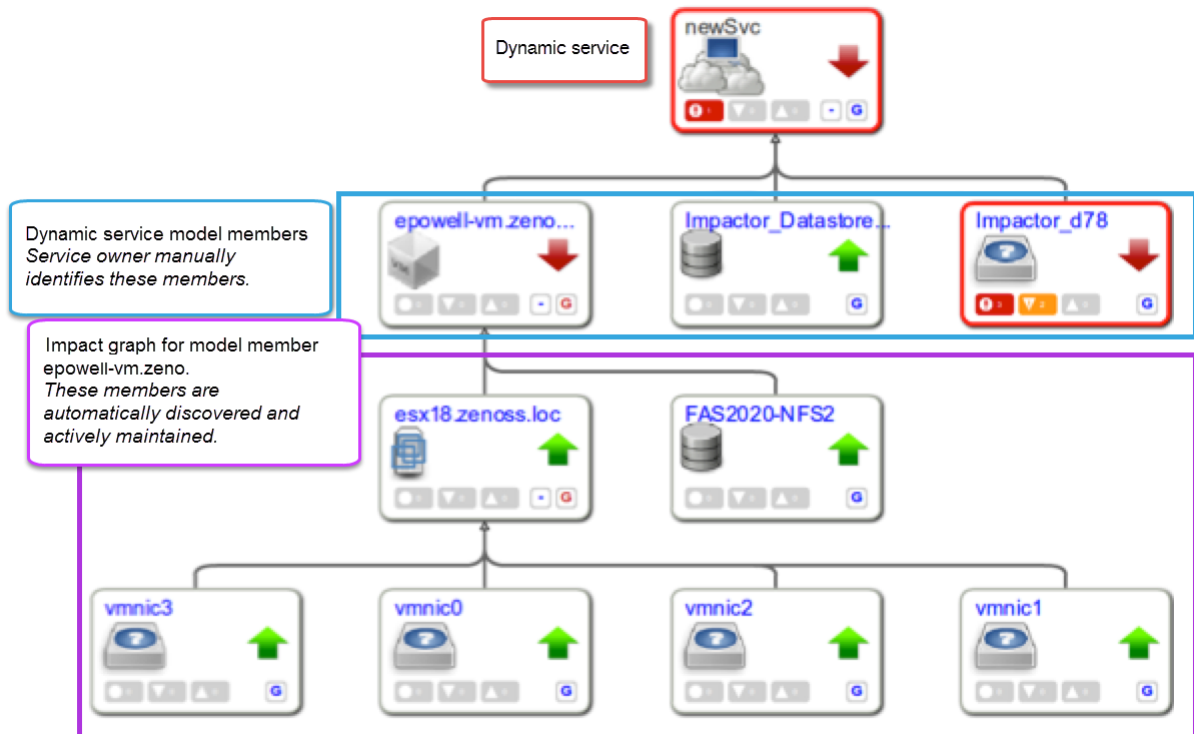


Service models, nodes, and graphs

With Service Impact, service modeling is simple. In your service model, you only define the main members of the service infrastructure, such as an application server and a database server. You do not need to specify or even be aware of the infrastructure entities on which the service model members depend.

Resource Manager automatically discovers and actively maintains the remaining infrastructure entities, such as hosts, operating systems, storage, network, compute, and other dependencies. In cloud and converged infrastructure environments, dependencies can be very complex and highly dynamic.

In the following figure, the top node of the graph is a the *service context (root) node*, which represents a dynamic service that you create. The second tier contains the child nodes of the root node, which represent the *service model members* that you identify when you define a dynamic service. In tiers three to *n* are the service model member's descendant nodes, the supporting infrastructure entities that are automatically discovered and maintained.



Service model members can be devices, components, component groups, logical nodes, organizing groups, or even other dynamic services, such as `Impactor_d78` in the figure.

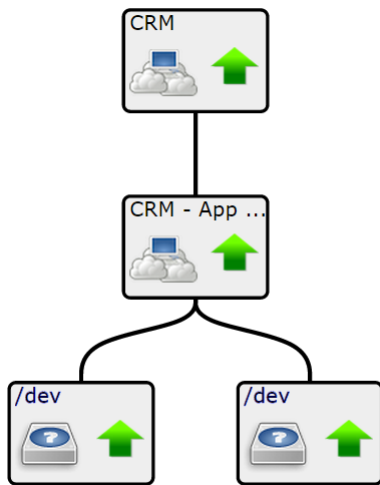
- *Device nodes* represent resources that are monitored by Resource Manager, such as a Linux server. Additionally, ZenPacks that model a device can extend the model to include other members. (For more information about Service Impact ZenPacks, contact Zenoss Support.)

Note Resource Manager provides the ability to reidentify devices. The reidentification process deletes and re-adds the device with a new ID and GUID. When the GUID is changed, the device is removed from that particular service. If you reidentify a device, you must manually re-add the device to the service model.

- *Component nodes* represent resources that are monitored by Resource Manager. Components and component groups usually represent device entities, such as an Ethernet port.
- *Logical nodes* are customizable and serve as filters against Resource Manager or external event sources. For events that match the logical node filter, the logical node identifies the state as a "placeholder" for the semantic meaning that is derived from the event filter definition. For example, events from an external application performance management (APM) source might be filtered to match events with a transaction that represents the health of the service. If events occur for that transaction, the logical node represents that negative state in the dynamic Service Impact graph, and propagates the impact as if the logical node were a device.
- *Organizing group nodes* represent the current definition of an organizing group's devices and all child organizing group hierarchies. For more information about organizing groups, see [Organizers and organizing groups](#) on page 12.

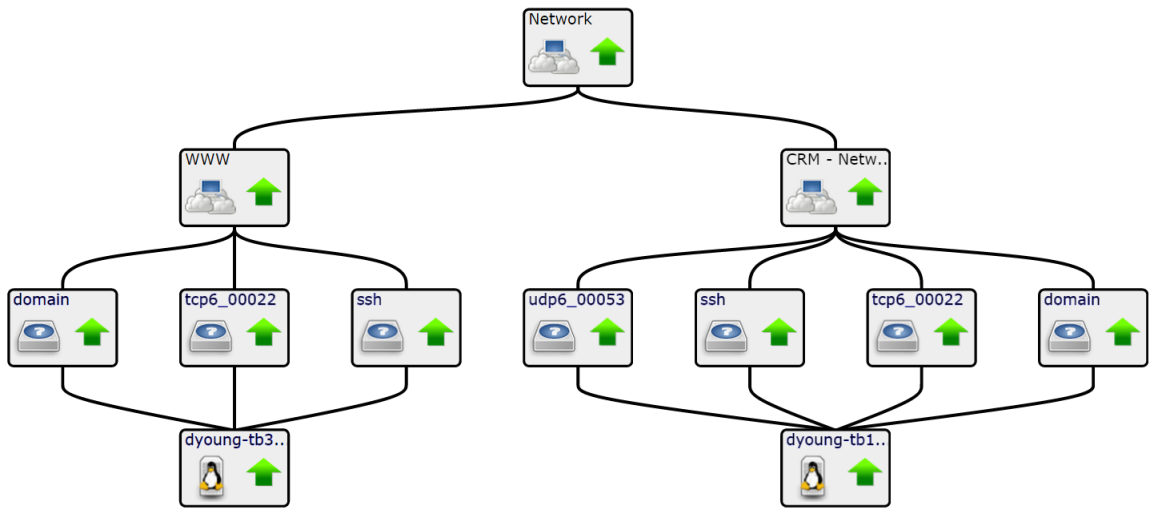
Service models can consist of a single monitored resource and its child dependencies. In the following example, the top node, CRM, is the dynamic service. The second node, CRM-App, is a service model member. The third tier, (`/dev` and `/dev`) are descendents of the CRM-App node.

Figure 2: Service model: Single monitored resource

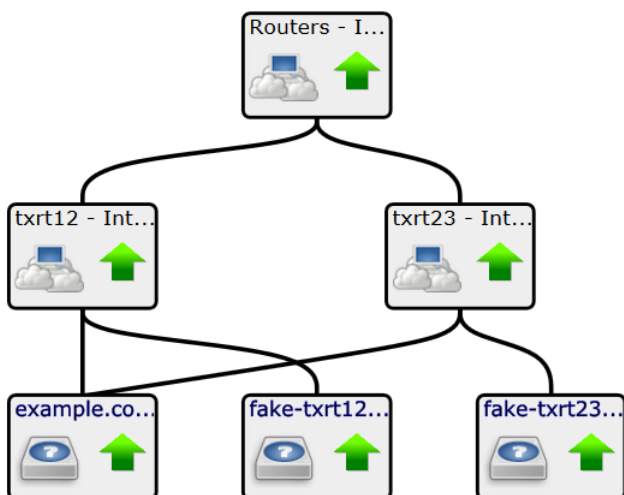


Service models can consist of multiple service models that are connected in a hierarchy to show their interdependencies. In the following example, the top tier, Network, and the second tier, WWW and CRM Netw nodes, are all dynamic services.

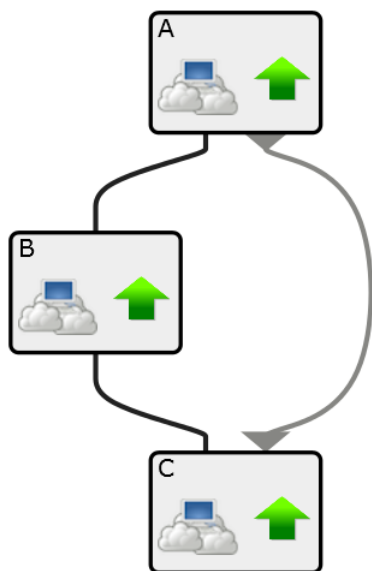
Figure 3: Service model: Hierarchy of service models



A node can be used in multiple *dynamic service contexts* as a service model member or as an infrastructure entity node to other service model members. For example, network-related nodes, such as switches or routers, are usually shared and appear in multiple service models in the service model hierarchy. In the following example, the node `example` is shared by `txrt12` and `txrt23`.

Figure 4: Dynamic service context: Shared nodes

A node can also exist as both a parent and a child within the same service model, creating a cyclic dependency. In the following example, node A is both a parent and a child in the service model, as indicated by the lighter arrow on the right.

Figure 5: Node with cyclic dependency

Dynamic service context

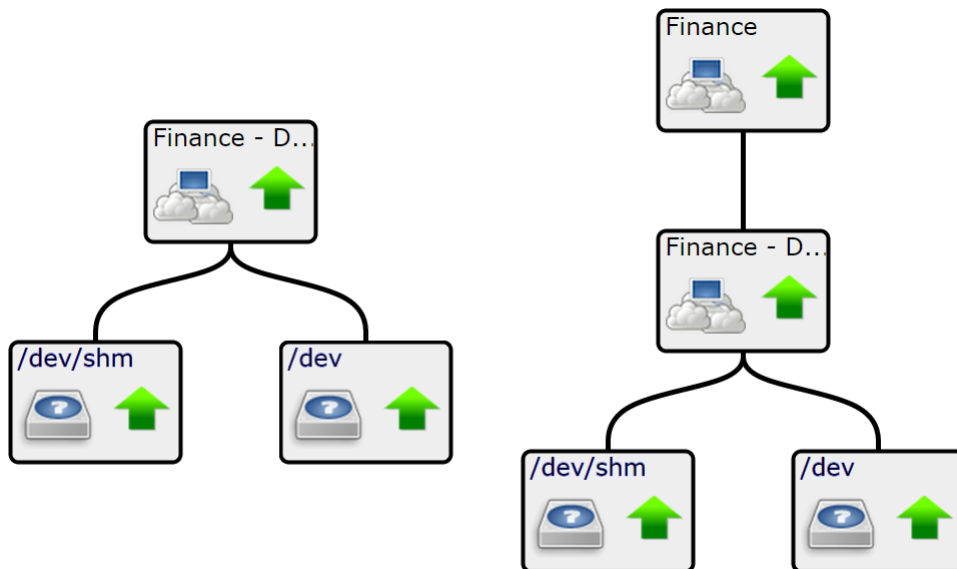
A dynamic service provides a *dynamic service context*, which includes the service itself, its service model members, and the impact graph for each of the service model members.

The service context also provides event propagation policies that identify whether and how events are propagated through the service impact graph. An event that occurs on a device member that participates in two service contexts might change its state and propagate the event in one service model's context, but behave differently in the other service context. For more information, see [State propagation policies](#) on page 13.

The *active service context* is the one that you are viewing or managing. It is always the highest-level member in the service model and node in the Service Impact graph. As you view or manage various service model members, be aware of their current service context.

The following example shows that service model members can be defined by other services, and that models can share common service members; however, because they are in different service models, they can have different service contexts.

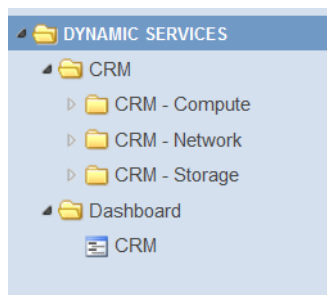
The service context of the service model on the left is Finance - Databases, and the service model contains two file systems. The service model on the right has service context Finance, and the service model contains the first service as its only service model member.



Organizers and organizing groups

The Resource Manager browser interface provides organizer folders for dynamic services and devices.

On the Resource Manager **SERVICES** page, organizers are available for grouping dynamic services. The organizers provide valuable context in service events and in Zenoss Analytics reports. The follow example shows Compute, Network, and Storage organizers for the CRM dynamic service.

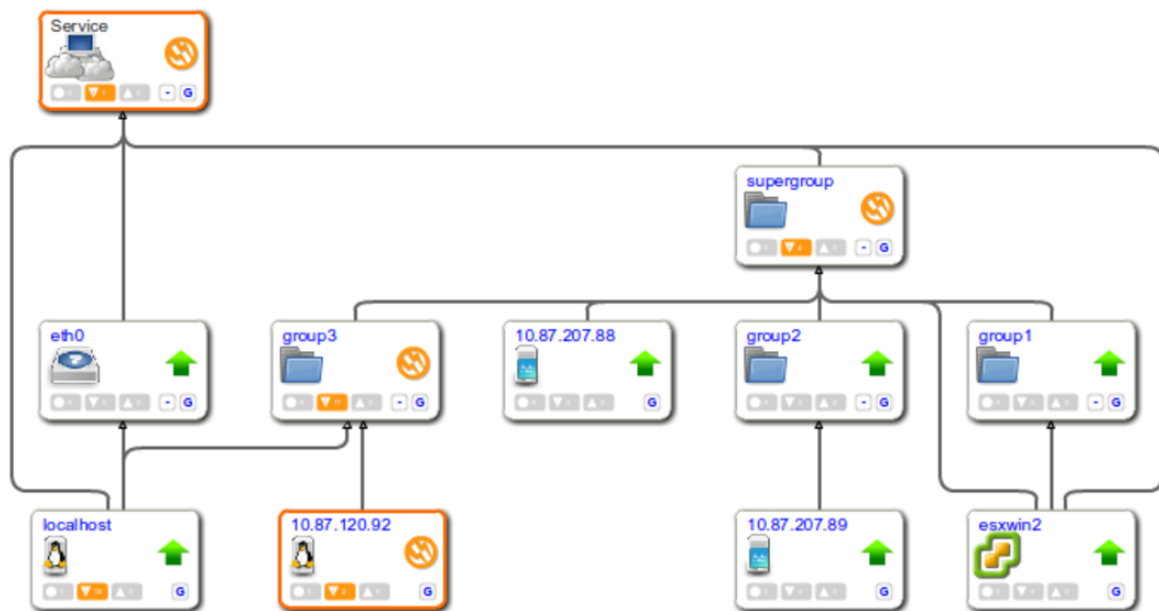


On the Resource Manager **INFRASTRUCTURE** page, organizing groups for groups, systems, and locations are available to organize devices into different categories. For example, to efficiently manage devices and use them in Service Impact, you might organize devices into groups based on their organizational group, system type, or physical location. The following example shows organizing groups.



You can add organizing groups to a service model for a dynamic service. As you add and remove devices and other organizing groups on the **INFRASTRUCTURE** page, the related service models and the impact graph for the organizing group automatically reflect the changes.

Figure 6: Service model with organizing groups



State propagation policies

State propagation policies for a dynamic service context determine how the Availability and Performance state of an entity's node and the nodes it impacts change because of events being received to that entity.

When new events occur against a service model member (device, component, component group, logical node, organizing group, or dynamic service) Service Impact identifies all of the service contexts in which that member participates. For each service context, Service Impact performs the following processing:

- Applies the combination of global, service, and node context policies to decide the state of that member in this service context.
- If the combination of policies indicate that the event should be propagated further within the service context, looks up all impact dependency relationships that exist for other nodes in the same service context. Different relationships can exist in other service contexts.

- Applies the service context policies to each of those nodes to compute their new state in the service context and decide whether to continue propagating further. Propagation of state change and event continues until all state and event propagations are completed.
- If any path through the impact graph affects the dynamic service itself, generates a service event. The service event includes details about the path from the event to the service model member through the impact graph to reach the dynamic service. Multiple paths through the graph from a single member's event might reach the dynamic service node, and multiple concurrent events to different members. Therefore, details of a service event include the union list of all service event paths and a probability ranking of which path and originating member event is the root cause.

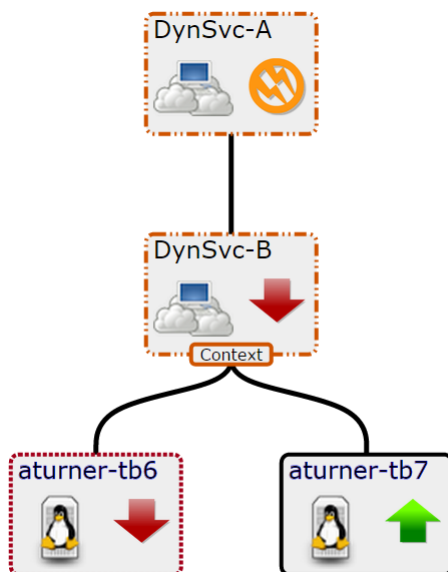
For a service model member that you use only to group child members, you can suppress sending service events.

You can create and assign multiple policies to any service member. In the **Impact View**, to select the type of policy that you want to create, edit, or view, select the Availability aspect or the Performance aspect.

The policy type that is applied to the members determines which members receive the data. The policy types, in order of precedence are as follows. For more information about state symbols and borders that are shown in the examples, see [Actual and derived state](#) on page 16.

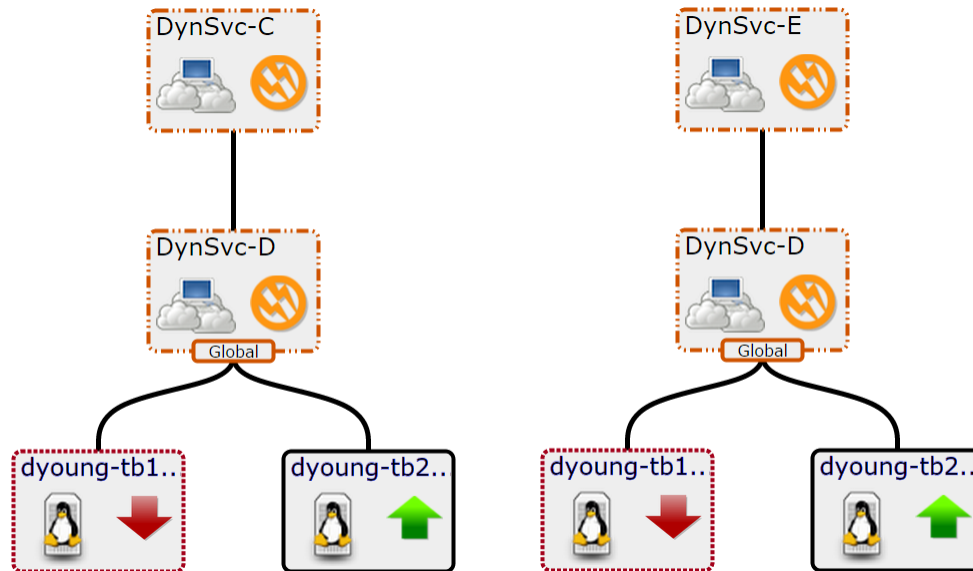
- 1 A *contextual policy* propagates the member's state change only to its immediate parent members within the current service context.

In the following example graph, a contextual policy is applied to the service model for Dynamic Service B (DynSvc-B). The policy states that if one child member is down, then the state of DynSvc-B must be degraded.



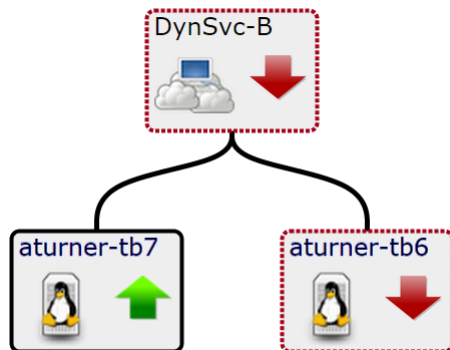
- 2 A *global policy* applies to all service model contexts that share a member that has a changed state. For example, if a global policy is applied to a member in a service model, and a child member has a change to its state data, the new state is propagated to the parent members in all service models to which the member belongs.

The following example graph shows that DynSvc-D is used in two different service models: DynSvc-C and DynSvc-E. The global policy propagates the degraded state to all service contexts of which DynSvc-D is a member.



- 3 A member with the *default policy* sends the state data of the worst condition affecting it to its parent member. The default policy is negated if you add either a contextual or global policy to a service model.

In the following example graph, no policy is applied to DynSvc-B. Its state is down because that is the worst case of its children.



If multiple policies are applied to a member, then the following overrides occur:

- A contextual policy overrides a global policy.
- Contextual and global policies override the default policy.

State triggers

A trigger defines a condition that the member's children must reach before the state change is propagated. For contextual policies, a state change is propagated to the parent members of the current context. For a global policy, a state change is propagated to the parent members of all service models with the same child member.

A policy includes one or more user-defined *state triggers*. If a policy contains multiple triggers, each trigger is evaluated before the state data is sent.

Members with *custom state providers* provide specialized options for defining state triggers. Service Impact always collects information from events that belong to most `/Status` subclasses, and all `/Perf` classes. You

can customize Resource Manager to gather state data from events belonging to other classes. For example, you can customize state providers to define state triggers for device and component members that are monitored through customized classes provided by the ZenVMware and CiscoUCS ZenPacks.

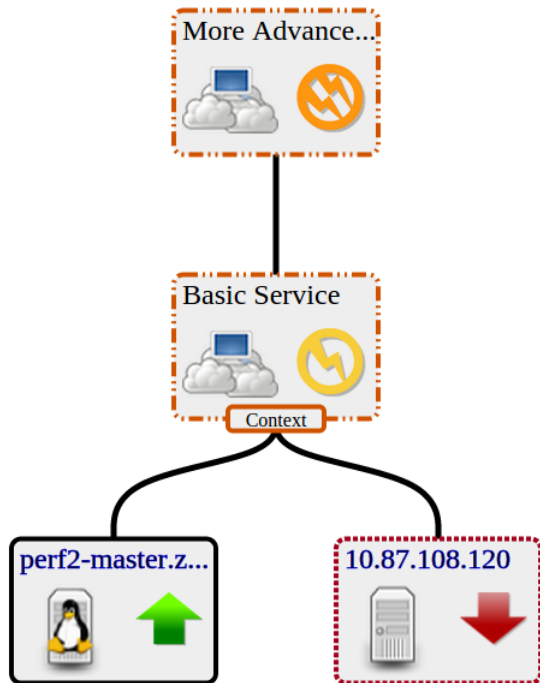
Note If you create a policy with a state trigger that cannot be met, a state change will go undetected and unreported, leaving a service model in an inaccurate state. For example, you might create a policy and define the state trigger to be met when two child members go down. However, if the service model that is applied to the policy contains only a single child member, the trigger will never be met and the state data will always be shown as UP. Because you applied a policy to the member, the default policy is also negated.

Actual and derived state

Each in a service model has an *actual state* and a *derived state*.


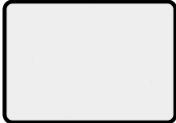





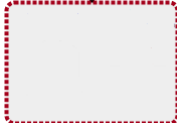
A device element's actual state is determined by events that occur on the device, regardless of the service models in which it participates. A service model member's actual state is generated from the service model in its own service context. The element's derived state is generated from policy propagation within a given context. In the Service Impact graph, the symbol that appears inside the node's border reflects the actual state; the border that outlines the node reflects the derived state.

In the following example graph, the actual and derived states are down. The parents of the device have a degraded and at-risk actual state, yet the derived state based on the contextual policy, is degraded.



Visual state indicators

The **Impact View** provides symbols and borders to visually indicate availability, actual state, performance, and derived state, as shown in the following table.

Availability	Performance	Symbol / Actual State	Border / Derived State
UP	ACCEPTABLE		
ATRISK	Not applicable.		
DEGRADED	DEGRADED		
DOWN	UNACCEPTABLE		

A member's actual and derived states can be different. As shown in the following example, a service's actual state can be up, but an applied context policy changed its derived state to down.



How policies work

The following example outlines the Service Impact policy information work-flow using a monitored switch as a device member.

- 1 A switch that is monitored by Resource Manager goes down.
- 2 Resource Manager generates an event about the switch.
- 3 Service Impact reads the switch-down event and changes the actual state of the device member that represents the switch.
- 4 For each service context in which the device member participates, Service Impact evaluates the state triggers to identify that member's derived state within the service context.
- 5 Service Impact propagates state data according to the context or global policies that are defined for the parent members of the device member.
- 6 If the device member's state causes a change in the derived state for the top-level member in the member's service context, Service Impact generates a service event and sends it to Resource Manager. (You can customize service event notifications.)
- 7 Service Impact performs a RCA and generates confidence rankings of how the switch-down event affects each service model to which the switch belongs.

Service model definition process

The key to defining accurate service models is thorough dependency discovery. Environmental factors influence the process of defining service models, such as the level of

- organizational maturity (processes and tooling)
- automation service life cycles (provisioning and management)
- application and infrastructure standardization

Despite these factors, the following steps outline a repeatable process for defining accurate service models.

1 Define the service to model.

In Service Impact graphs, a service and its members are represented as *service nodes*. A service is defined by its boundaries and type.

- A web service (for example, a human resources portal) or an IaaS platform (an Amazon EC2 instance) depends on specific internal or external resources.
- Software as a service (for example, *ServiceNow*) is defined by its deployment architecture.

2 Define service models for subservices.

In Service Impact graphs, subservices are represented as *service nodes*. (Service node roles are contextual.) A subservice has a direct relationship with one or more services. If the subservice fails or degrades, the service fails or degrades based on the propagation rules for that service context.

Often, a subservice represents an infrastructure tier, such as a gateway or database service. At a lower level, a tier might be configured with redundant members for high availability. However, Zenoss recommends modeling subservices as single points of failure.

- 3 Add device nodes, component nodes, logical nodes, and organizing groups to service and subservice nodes. After you define subservices, you can add the infrastructure resources that make up the subservices.
- 4 Define global policies on subservices. Policies capture domain knowledge about services, and enable the automated dependency tracking and RCA computation that make Service Impact so valuable. To facilitate re-use of subservices, use global policies instead of contextual policies wherever possible.

Contextual policies are valuable, but their use cases are relatively rare because most deployment scenarios use shared resources. For example, a hypervisor hosts virtual machines that belong to separate services. One service member with a global policy can apply to all virtual machines, across all service models. The service relationship, not service ownership, is the key to determining the relevance of events and sending state data to parent members.

5 Repeat steps 2 - 4 as needed.

6 To identify gaps in monitoring processes, analyze failure scenarios.

Are key measurement points of each member in the service model properly monitored? For example, are synthetic transactions in place for web servers? Are ping checks being performed against host operating systems? Service Impact can only act on events that flow through Resource Manager.

7 Test failure scenarios.

The `zensendevent` command generates synthetic events, which are invaluable for validating service relationships, policies, and even monitoring functions, before real events or event storms occur. For more information about `zensendevent`, refer to the *Zenoss Resource Manager Administration Guide*.

8 Refine the service model.

If a test or gap analysis reveals missing policies or service members, add them and test again.

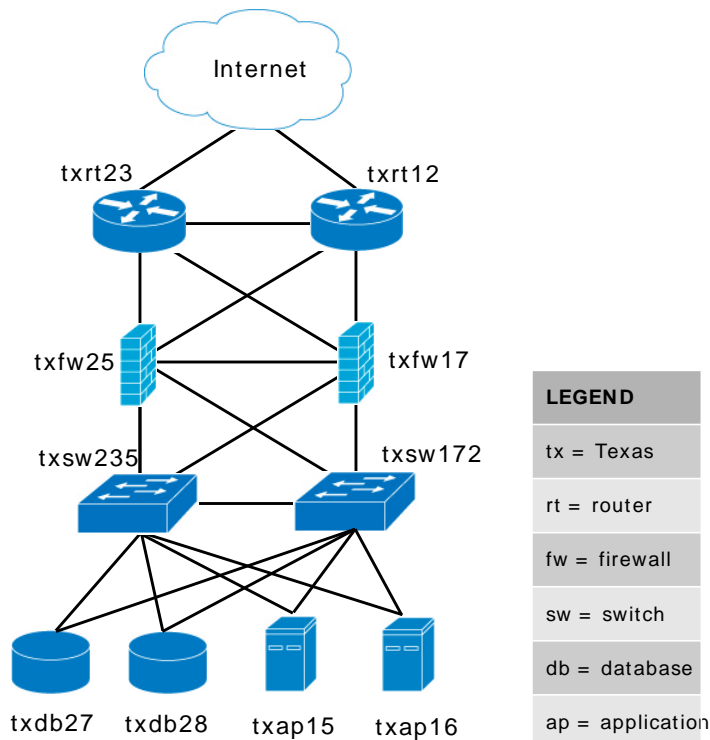
Tutorial: Creating a service model

This tutorial demonstrates how to create a service model for a simplified CRM application, view the model, and view and interpret the root-cause analyses that Service Impact creates when events affect the application's availability state.

About this tutorial:

- For information about Service Impact browser interface elements, see [Browser interface resources](#) on page 47.
- Complete the tasks in the order in which they are presented.
- Where indicated, steps depend on the version of Resource Manager with which Service Impact is deployed.
- Device and component resources for this tutorial are fake, and do not affect production environments. The final task erases the fake devices and the nodes that you create during the tutorial. For more information, see [Tutorial devices and member types](#) on page 21.

The following diagram shows the network topology of the CRM application that is featured in this tutorial.



Define the service to model

The service to model is a CRM application, and it is defined by the members (resources) on which it relies.

The CRM application relies on application and database hosts and processes, and on the network infrastructure that connects the service with its users. This tutorial creates a service model of the development deployment of the CRM application, not the production or quality assurance deployments.

The following procedure loads the fake devices into Resource Manager and sets up the tutorial environment.

- 1 *(Resource Manager 5.x and later)* Gain access to the Resource Manager CLI environment in a Zope container.
 - a Log in to the Control Center master host as a user with serviced CLI privileges.
 - b Start an interactive session in a Zope container as user zenoss.

```
serviced service attach zope/0 su - zenoss
```

- 2 *(Resource Manager 4.2.x)* Log in to the Resource Manager master host as zenoss.
- 3 Create a soft link to the Service Impact scripts directory.
 - a Create a variable for the path of the ZenPacks.zenoss.Impact ZenPack.

```
my_zp=$(zenpack --list | awk '/\\.Impact- / \
{ print substr($2,2,length($2)-2) }')
```

- b Create the link.

```
ln -s $my_zp/ZenPacks/zenoss/Impact/scripts/tutorial \
${HOME}/impact_scripts
```

- 4 Change directory to the Service Impact scripts directory.

```
cd ${HOME}/impact_scripts
```

- 5 Load fake devices and components into Resource Manager.

```
zenbatchload --nomodel ./devices.txt
```

- 6 In `tutorial.sh`, ensure that the values of the `ZENOSS_USERNAME` and `ZENOSS_PASSWORD` variables specify the user name and password of a Resource Manager user account, and if not, change them.
- 7 Set up the tutorial environment.

```
./tutorial.sh
```

- 8 Log in to the Resource Manager browser interface as a user with ZenManager or Manager privileges.
- 9 Click **SERVICES**.

The Service Impact feature of Resource Manager adds a tab named **SERVICES** to the Resource Manager menu bar.

- 10 In the tree view, open the **CRM - Development** organizer, and then open the **Application** and **Compute** organizers.

The tree view displays the service members created by the `tutorial.sh` script, as shown in the following example.

Service	Availability	Service	Performance
Apache Daemons	UP	Apache Daemons	ACCEPTABLE
App Hosts	UP	App Hosts	ACCEPTABLE
CRM - Application Service	UP	CRM - Application Service	ACCEPTABLE
CRM - Compute Service	UP	CRM - Compute Service	ACCEPTABLE
DB Hosts	UP	DB Hosts	ACCEPTABLE
Java Daemons	UP	Java Daemons	ACCEPTABLE
MySQL Daemons	UP	MySQL Daemons	ACCEPTABLE
bfw17 - Routers	UP	bfw17 - Routers	ACCEPTABLE
bfw25 - Routers	UP	bfw25 - Routers	ACCEPTABLE
bfw172 - App Hosts	UP	bfw172 - App Hosts	ACCEPTABLE
bfw172 - DB Hosts	UP	bfw172 - DB Hosts	ACCEPTABLE
bfw172 - Firewalls	UP	bfw172 - Firewalls	ACCEPTABLE
bfw235 - App Hosts	UP	bfw235 - App Hosts	ACCEPTABLE
bfw235 - DB Hosts	UP	bfw235 - DB Hosts	ACCEPTABLE
bfw235 - Firewalls	UP	bfw235 - Firewalls	ACCEPTABLE
zfake bfw17 - bxt12	UP	zfake bfw17 - bxt12	ACCEPTABLE
zfake bfw17 - bxt23	UP	zfake bfw17 - bxt23	ACCEPTABLE
zfake bfw25 - bxt12	UP	zfake bfw25 - bxt12	ACCEPTABLE

Tutorial devices and member types

The tutorial uses the following fake devices and service model member types (Resource Manager device or component).

Application host 15

Device name	Description	Member type
fake-txap15	Application host 15	Device
fake-txap15-httpd	Apache daemon	Component
fake-txap15-java	Java/JRE daemon	Component
fake-txap15-nic-0	Network interface card 0	Component
fake-txap15-nic-1	Network interface card 1	Component

Application host 16

Device name	Description	Member type
fake-txap16	Application host 16	Device
fake-txap16-httpd	Apache daemon	Component
fake-txap16-java	Java/JRE daemon	Component
fake-txap16-nic-0	Network interface card 0	Component
fake-txap16-nic-1	Network interface card 1	Component

Database host 27

Device name	Description	Member type
fake-txdb27	Database host 27	Device
fake-txdb27-mysqld	MySQL daemon	Component
fake-txdb27-nic-0	Network interface card 0	Component
fake-txdb27-nic-1	Network interface card 1	Component

Database host 28

Device name	Description	Member type
fake-txdb28	Database host 28	Device
fake-txdb28-mysqld	MySQL daemon	Component
fake-txdb28-nic-0	Network interface card 0	Component
fake-txdb28-nic-1	Network interface card 1	Component

Firewall 17

Device name	Description	Member type
fake-txfw17	Firewall 17	Device
fake-txfw17-10g-0	Port 0 (10GB capacity)	Component
fake-txfw17-10g-1	Port 1 (10GB capacity)	Component
fake-txfw17-10g-2	Port 2 (10GB capacity)	Component

Device name	Description	Member type
fake-txfw17-10g-3	Port 3 (10GB capacity)	Component
fake-txfw17-1g-0	Port 0 (1 GB capacity)	Component

Firewall 25

Device name	Description	Member type
fake-txfw25	Firewall 25	Device
fake-txfw25-10g-0	Port 0 (10GB capacity)	Component
fake-txfw25-10g-1	Port 1 (10GB capacity)	Component
fake-txfw25-10g-2	Port 2 (10GB capacity)	Component
fake-txfw25-10g-3	Port 3 (10GB capacity)	Component
fake-txfw25-1g-0	Port 0 (1 GB capacity)	Component

Router 12

Device name	Description	Member type
fake-txrt12	Router 12	Device
fake-txrt12-100g-0	Port 0 (100GB capacity)	Component
fake-txrt12-10g-0	Port 0 (10GB capacity)	Component
fake-txrt12-10g-1	Port 1 (10GB capacity)	Component
fake-txrt12-1g-0	Port 0 (1GB capacity)	Component

Router 23

Device name	Description	Member type
fake-txrt23	Router 23	Device
fake-txrt23-100g-0	Port 0 (100GB capacity)	Component
fake-txrt23-10g-0	Port 0 (10GB capacity)	Component
fake-txrt23-10g-1	Port 1 (10GB capacity)	Component
fake-txrt23-1g-0	Port 0 (1GB capacity)	Component

Switch 172

Device name	Description	Member type
fake-txsw172	Switch 172	Device
fake-txsw172-10g-0	Port 0 (10GB capacity)	Component
fake-txsw172-10g-1	Port 1 (10GB capacity)	Component
fake-txsw172-1g-0	Port 0 (1GB capacity)	Component
fake-txsw172-1g-1	Port 1 (1GB capacity)	Component

Device name	Description	Member type
fake-txsw172-1g-2	Port 2 (1GB capacity)	Component
fake-txsw172-1g-3	Port 3 (1GB capacity)	Component
fake-txsw172-1g-4	Port 4 (1GB capacity)	Component

Switch 235

Device name	Description	Member type
fake-txsw235	Switch 235	Device
fake-txsw235-10g-0	Port 0 (10GB capacity)	Component
fake-txsw235-10g-1	Port 1 (10GB capacity)	Component
fake-txsw235-1g-0	Port 0 (1GB capacity)	Component
fake-txsw235-1g-1	Port 1 (1GB capacity)	Component
fake-txsw235-1g-2	Port 2 (1GB capacity)	Component
fake-txsw235-1g-3	Port 3 (1GB capacity)	Component
fake-txsw235-1g-4	Port 4 (1GB capacity)	Component

Introduction to the tutorial environment

The `tutorial.sh` script creates the following service model members and organizers to initialize the CRM service model in Service Impact. Use of these members and organizers is a recommended best practice.

- **Dashboard** organizer; root-level organizer for service models as a whole. Initially, the organizer is empty.
- Root-level **CRM - Development** organizer that contains additional organizers. Use a single root-level organizer to contain the subservices of each service model, and use standardized names (and contents) for sub-organizers.
- **CRM - Application Service** and **CRM - Compute Service** service model members, children of the **CRM - Development** organizer.

These members summarize the application and compute services that are associated with the CRM application. They are easily located without having to open the organizers in which their constituent subservices are located.

- Members in the **Network** organizer that start with **zfake** represent the network connections between fake devices.

Because these fake devices are not modeled, Resource Manager cannot discern their relationships, and Service Impact cannot create device or component members. Therefore, the script creates members to represent the connections.

These members have neither contextual nor global policies, so the default policy applies. That is, the state of the worst condition that affects child members becomes the state of the **zfake** members, which is the correct policy for these connections.

- DNS and interface names that follow a naming convention.
- Subservice members in the **Network** organizer that start with **tx**, which contain redundant resources. Standardized, global availability policies are defined.

These subservice members demonstrate the following **best practices**:

- Each subservice member contains homogeneous child members. Global policies work best when child members are homogeneous.
- Each subservice uses global policies. Global policies can be re-used across service model boundaries. Contextual policies are restricted to specific service models.
- Each global policy contains the following, standardized availability state triggers: ATRISK if 50% or more child members are down; DOWN if 100% of child members are down. Use of percentage thresholds means that the policies do not need to be adjusted if additional resources are deployed later.

Note The standardized state triggers that are used in this case are examples of systematically thinking about and using global policies, and not intrinsically best practices. For example, if a resource pool contains more than two members, additional state triggers can be defined.

The remaining procedures in this tutorial demonstrate how to complete and test the CRM service model.

Create logical node and subservice members for Internet connections

All network connections except the connection to the Internet are modeled. Because the Internet connection is not modeled in Resource Manager, we create a logical node to represent it. For this tutorial, the logical node is simply a service model member that reacts to fake events. A "real" logical node could be configured to react to events from a `zencommand` that pings an Internet resource.

Create logical node

- 1 In the Resource Manager browser interface, select **SERVICES > Logical Nodes**.
- 2 From the **Add** menu at the bottom of the tree view, select **Add Logical Node Organizer**.

- 3 In the **Add Logical Node Organizer** dialog box, enter `CRM - Development`, and then click **SUBMIT**.
- 4 From the **Add** menu at the bottom of the tree view, select **Add Logical Node**.
- 5 In the **Add Logical Node** dialog box, enter `example.com`, and then click **SUBMIT**.
- 6 In the `example.com` details view, enter values to match the following table, and then click **Save**.

Field	Description
Description	A node to help represent a route to the Internet.
Criteria	all, Summary, contains, fakeInternet
Events for this node in this event class	/Status/Ping
will result in these availability states	Critical: DOWN, Error: DOWN, Warning: ATRISK, Clear: UP

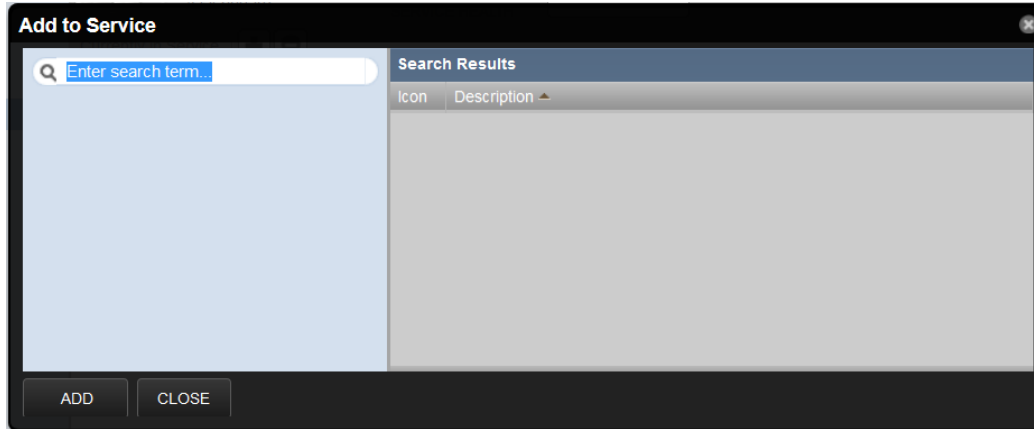
Create a subservice member for one Internet connection

Create a service model member for the connection from router 12 to the `zenoss.com` logical node.

- 1 In the Resource Manager browser interface, select **SERVICES > Dynamic Services**.
- 2 In the tree view of organizers, open **CRM - Development > Network**.

Unless an organizer is selected, new members are created at the root level. From there, you can drag the new member into the correct organizer.

- 3 From the **Add** menu at the bottom of the tree view, select **Add Dynamic Service**.
- 4 In the **Add Dynamic Service** dialog box, enter `txrt12 - Internet`, and then click **SUBMIT**.
- 5 In the **Members** view, click **Add**.
- 6 In the **Add to Service** search field, enter `fake-txrt12`.



- 7 In the left column, select **Device**, and in the results list, select `fake-txrt12-100g-0`, and then click **ADD**.
- 8 In the search field, enter `example.com`.
- 9 From the search results list, select `example.com`, and then click **ADD > CLOSE**.

Create a subservice member for the other Internet connection

To create the service model member for the connection through router 23, clone the member for the connection through router 12.

- 1 In the tree view, select service model member `txrt12 - Internet`.
- 2 From the **Action** menu at the bottom of the tree view, select **Clone Service**.
- 3 In the **Clone Service** dialog box, enter `txrt23 - Internet`, and then click **SUBMIT**.
The new service is created and its contents are displayed in the **Members** view.
- 4 From the list of members in the **Members** view, select `fake-txrt12-100g-0`, and then click **Remove**.
If you click on the name of the member, Resource Manager displays the device overview page. To return to the correct page, click the browser's back button.
- 5 In the **Remove Items** dialog box, click **OK**.
- 6 In the **Members** view, click **Add**.
- 7 In the **Add to Service** search field, enter `100g`.
- 8 In the left column, select **Device**.
- 9 In the results list, double-click `fake-txrt23-100g-0`, and then click **ADD > CLOSE**.

Create service member to represent redundant paths to the Internet

The previous tasks created subservices to represent the WAN connections to the Internet. This task creates a subservice member in the service model to represent the redundant WAN tier.

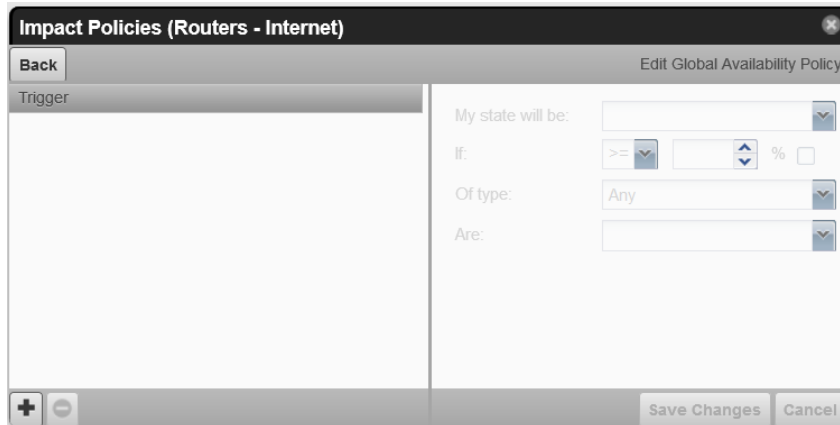
- 1 In the tree view, select **Network**.
- 2 From the **Add** menu, select **Add Dynamic Service**.
- 3 In the **Add Dynamic Service** dialog box, enter `Routers - Internet`, and then click **SUBMIT**.
- 4 In the **Members** view, click **Add**.
- 5 In the **Add to Service** search field, enter `Internet`.
- 6 In the left column, select **DynamicService**.

- 7 In the results list, select **txrt12-Internet** and **txrt23-Internet**, and then click **ADD > CLOSE**.

Add a policy to the Internet connection member

Add a global availability policy to the Internet connection subservice member.

- 1 In the tree view, select **Routers - Internet**.
- 2 From the view menu, select **Impact View**.
- 3 In the node tile of the **Routers - Internet** service (the tile at the top of the hierarchy), right-click and then select **Edit Impact Policies**.
- 4 In the **Impact Policies** dialog box, directly to the right of the **Global Policy** entry, click **Add**.



- 5 In the lower-left corner of the **Edit...Policy** tab, click **Add**, and then add the following triggers.
 - ATRISK if `>= 50% DynamicService` is DOWN
 - DOWN if `>= 100% DynamicService` is DOWN

Note When you select a type for a state trigger, the selection excludes other types. In this case, the only options are **Any** and **DynamicService**, and both child members are service members, so exclusivity is not a concern.

Create members for major network segments

All connections between devices and all redundant resources are modeled. This procedure creates service model members for the major network segments.

- 1 In the Resource Manager browser interface, select **SERVICES > Dynamic Services**.
- 2 In the tree view, open **CRM - Development > Network**.
- 3 Create new members for the major network segments. For instructions, see preceding steps.

The following table shows new and existing members.

New service model member	Existing service model members
App Hosts - Switches	txsw235 - App Hosts, txsw172 - App Hosts
DB Hosts - Switches	txsw235 - DB Hosts, txsw172 - DB Hosts
Firewalls - Routers	txfw25 - Routers, txfw17 - Routers
Switches - Firewalls	txsw235 - Firewalls, txsw172 - Firewalls

- 4 Add the following global policies (availability state triggers) to each new service model member. For instructions, see preceding steps.
 - `ATRISK if >= 50% DynamicService is DOWN`
 - `DOWN if >= 100% DynamicService is DOWN`

Create summary members for critical paths

Service model members for all of the major network segments are in place. Now consider members to represent the critical paths. To characterize the CRM application as available, the following minimum connections are required:

- One connection between Internet users and an application server that is UP.
- One connection between an application server and a database server that is UP.

If entities for these paths existed, we could create the service model member that summarizes the network service for CRM. However, only one segment is defined, by the **Routers - Internet** subservice. The following paths require an additional subservice each:

- The path between the routers and the application hosts.
- The path between the application and database hosts.

- 1 In the Resource Manager browser interface, select **SERVICES > Dynamic Services**.
- 2 In the tree view, open the **CRM - Development** organizer, and then open and select the **Network** organizer.
- 3 Create new service model members for the network paths that affect users.

Because each subservice is critical, the correct policy for these new service model members is the default policy. The following table matches new and existing service model members.

New service model member	Existing service model members
Routers - App Hosts	App Hosts - Switches, Firewalls - Routers, Switches - Firewalls
App Hosts - DB Hosts	App Hosts - Switches, DB Hosts - Switches

For detailed instructions, see the preceding steps.

Create a member for the network service

The critical network paths are defined. Create a service model member to represent the network service for the CRM application.

- 1 In the Resource Manager browser interface, select **SERVICES > Dynamic Services**.
- 2 In the tree view, open the **CRM - Development** organizer, and then select it. Service model members that represent a category should be peers of their sub-organizers.
- 3 Create a new service model member, named `CRM - Network Service`.
- 4 Add the following subservice model members to the new service model member.
 - App Hosts - DB Hosts
 - Routers - App Hosts
 - Routers - Internet

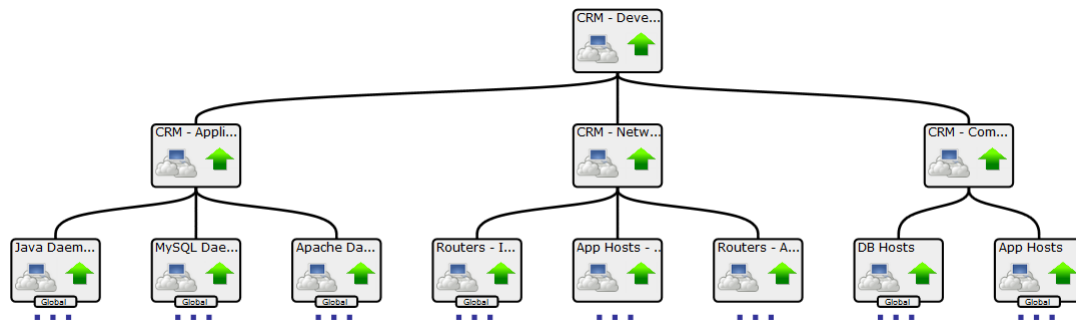
The correct policy for this member is the default policy.

Create a member for the service model

After all resource categories are modeled, the application as a whole can be modeled.

- 1 In the Resource Manager browser interface, select **SERVICES > Dynamic Services**.
- 2 In the tree view, select the **Dashboard** organizer.
Keep service model members that represent a service model as a whole in a separate, root-level organizer. This way, you can quickly determine the state of all service models in the environment.
- 3 Create a service model member named `CRM - Development Service` and add the following subservice model members to it.
 - CRM - Application Service
 - CRM - Compute Service
 - CRM - Network Service

To view the graph of the service model, select the new service in the tree view, and then select **Impact View**. After hiding child nodes and zooming in, the graph looks similar to the following image. If necessary, to reposition the graph, click and drag it.



Send events to fake devices

To see how they affect the availability of the CRM application, send events to the fake devices.

- 1 In the tree view, open the **Dashboard** organizer, and then select **CRM - Development Service**.
- 2 In the main view area, select **Impact View**.
- 3 (*Resource Manager 5.x and later*) Gain access to the Resource Manager CLI environment in the ZenHub container.
 - a Log in to the Control Center master host as a user with `serviced` CLI privileges.
 - b Start an interactive session in a Zope container.

```
serviced service attach zenhub
```

- c In the new session, switch user to `zenoss`.

```
su - zenoss
```

- 4 (*Resource Manager 4.2.x only*) Log in to the Resource Manager master host as `zenoss`.
- 5 Send ping down events to the fake network interface card components in the `txap15` and `txap16` hosts.

```
zensendevent -d fake-txap15-nic-0 -c /Status/Ping -s Critical \  
  "Impact Tutorial - fake device is DOWN"  
zensendevent -d fake-txap15-nic-1 -c /Status/Ping -s Critical \  
  "Impact Tutorial - fake device is DOWN"
```

```
zensendevent -d fake-txap16-nic-0 -c /Status/Ping -s Critical \
  "Impact Tutorial - fake device is DOWN"
```

- 6 In the browser, click **Refresh**.

The **Impact View** shows that the CRM application availability state changed to ATRISK, and the nodes involved in the state change are highlighted with yellow and red.

To see all nodes in the Service Impact graph, click **Expand All**.

- 7 Change the view to **Impact Events**, and then in the **Impact Events** list, click **CRM - Development Service**.

Status	Severity	Service	Event Class	Summary	First Seen	Last Seen	Count
!	Critical	CRM - Development Service	/Service/State/Availability	Service Dashboard/CRM - Development Service is ATRISK.	2013-10-24 12:37:05	2013-10-24 12:37:05	1

Confidence	Status	Severity	Resource	Component	Event Class	Summary	First Seen	Last Seen	Count
38	+	Critical	fake-txap16-n...	/Status/Pi...	/Status/Pi...	Impact Tutorial - fake device is DOWN	2013-10-24 12:37:05	2013-10-24 12:37:05	1
38	+	Critical	fake-txap15-n...	/Status/Pi...	/Status/Pi...	Impact Tutorial - fake device is DOWN	2013-10-24 12:37:04	2013-10-24 12:37:04	1
26	+	Critical	fake-txap15-n...	/Status/Pi...	/Status/Pi...	Impact Tutorial - fake device is DOWN	2013-10-24 12:37:04	2013-10-24 12:37:04	1

The events that contribute to the current state of the CRM application are weighted by the root-cause analysis that Service Impact performs (the **Confidence** column). To view the impact chain of an event, click the plus button in the left column.

- 8 Send ping down events to the fake network interface card components in the txdb27 and txdb28 hosts, and then click **Refresh**.

```
zensendevent -d fake-txdb27-nic-0 -c /Status/Ping -s Critical \
  "Impact Tutorial - fake device is DOWN"
zensendevent -d fake-txdb27-nic-1 -c /Status/Ping -s Critical \
  "Impact Tutorial - fake device is DOWN"
zensendevent -d fake-txdb28-nic-0 -c /Status/Ping -s Critical \
  "Impact Tutorial - fake device is DOWN"
```

The list of contributing events grows, and the rankings change to reflect the added events.

- 9 Send clear events to all fake devices that are down.

```
zensendevent -d fake-txap15-nic-0 -c /Status/Ping -s Clear \
  "Impact Tutorial - fake device is UP"
zensendevent -d fake-txap15-nic-1 -c /Status/Ping -s Clear \
  "Impact Tutorial - fake device is UP"
zensendevent -d fake-txap16-nic-0 -c /Status/Ping -s Clear \
  "Impact Tutorial - fake device is UP"
zensendevent -d fake-txdb27-nic-0 -c /Status/Ping -s Clear \
  "Impact Tutorial - fake device is UP"
zensendevent -d fake-txdb27-nic-1 -c /Status/Ping -s Clear \
  "Impact Tutorial - fake device is UP"
zensendevent -d fake-txdb28-nic-0 -c /Status/Ping -s Clear \
  "Impact Tutorial - fake device is UP"
```

Remove tutorial service model members

This step removes all tutorial-defined members.

- 1 Log in to the Resource Manager browser interface as a user with ZenManager or Manager privileges.
- 2 Click **SERVICES**.
- 3 Remove the dynamic services and logical node.
 - a In the tree view, select the **Dashboard** organizer.
 - b At the bottom of the tree view, click **Delete**.
 - c In the tree view, select **CRM - Development**.
 - d At the bottom of the tree view, click **Delete**.
 - e Select the **Logical Nodes** view mode.
 - f In the tree view, select **CRM - Development**.
 - g At the bottom of the tree view, click **Delete**.
- 4 Remove the fake devices.
 - a In the Resource Manager browser interface, select **INFRASTRUCTURE**.
 - b In the tree view, select **FakeImpactDevices**.
 - c At the bottom of the tree view, click **Delete**.

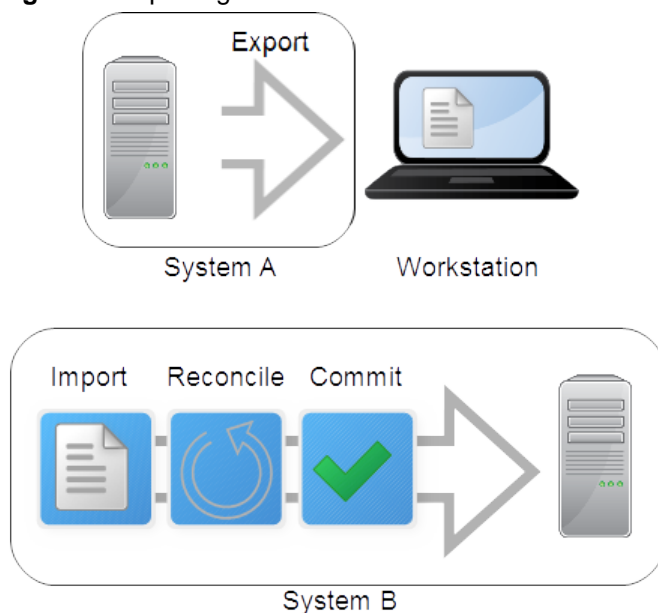
Exporting service models from one system to another

3

You can create a service model on one Resource Manager system (the originating system) and export that model to another system (the target system). This feature is useful for tasks such as repopulating a test or staging environment from a production environment, and performing repeated exporting and model synchronization from develop, to staging, to production environments. Review the following use case example and process overview.

"Project Zebra" is a complicated new E-commerce web application with many clustered entities. The application has been tested in a staging environment (the originating Resource Manager system), and the service model is ready to export from staging to the production environment (the target Resource Manager system). The following diagram illustrates the steps in the process; System A is the originating system and System B is the target system.

Figure 7: Exporting service models



Export overview

On the originating system, select the dynamic service, dynamic service organizer, or set of dynamic services to export their service models. The export process generates a file in *GraphML* XML format that describes the structural properties of the service models, including the devices, components, logical

nodes, and their relationships. On your workstation, save the exported file (referred to in this document as `svc_export.graphml`).

The export file contains definitions for entities that are explicitly defined in the service model. It does not include nodes that Service Impact identified as having an impact relationship and added. For example, service model SM1 defines only an application server device member, App Server A. However, Service Impact evaluated App Server A and infrastructure dependencies on the VM, VM OS, VM host, and file systems on which it resides. Service Impact automatically added the related service model members to the service model SM1. When service model SM1 is exported, the export file contains only service model SM1 and App Server A. When service model SM1 is imported and committed on the target system, a new version of service model SM1 is generated that is unique to the infrastructure dependencies of the target system.

Import overview

On the target system, you import the exported file, which imports the service model definitions to view into a working “sandbox” testing environment. While in the sandbox, service models are not operational and are not available in the Resource Manager browser interface. Before moving a service model out of the sandbox, you must complete the reconcile and commit steps.

The initial import process automatically attempts to reconcile (match) each service model member that the exported file references with a corresponding entity on the target system. For example, for Project Zebra, some devices in the staging and production environments might be shared. Others, like the database, will be replaced in the production environment by different entities in the same role. With the higher scalability that production requires, the clusters in the production environment might have more entities. Some entities might not exist or be known on the target system. Such differences must be reconciled before the new application is moved into production.

The import process generates text files to report the results and identify UNRECONCILED entities. This document refers to the files as `svc_export.graphml.latest.txt` and `svc_export.graphml.nnnn.txt`.

Reconcile overview

The generated text files provide known details about UNRECONCILED entities. Using that information, you search for and edit `svc_export.graphml.latest.txt` to identify corresponding entities on the target system. Your edited version of `svc_export.graphml.latest.txt` provides input for each reconcile attempt.

Commit overview

After you have reconciled the service models in the sandbox, perform the final commit. The committed service models are active and begin analyzing Resource Manager events when appropriate, and generating service events. View and work with the service models in the **SERVICES** tab of the Resource Manager browser interface.

Details about entities that required manual reconciliation are retained on the target system. Future service model imports of the same entities will be reconciled automatically. For example, for Project Zebra, the first import-reconcile-commit process on System B required a manual mapping of database “Test_DB1” to “Prodxn_DB1.” Future imports on System B of service models that reference “Test_DB1” will automatically refer to “Prodxn_DB1” instead.

You cannot reverse a commit; however, you can start a new import process or edit the service model definition in the Resource Manager browser interface.

Exporting a service model

Select service models on the originating system and generate the export file that you will use as input to import on the target system.

- 1 In the Resource Manager browser interface, select the **SERVICES** tab.
- 2 In the tree view, select the service models to export.
- 3 From the **Action** menu at the bottom of the tree view, select **Export Selected**.
- 4 Save the generated export file (`svc_export.graphml`) on your workstation.
- 5 Proceed with [Importing service model definitions](#) on page 34.

Importing service model definitions

Perform the procedures in this section to import a service export file in graphml format that contains service model definitions.

The service models in the export file are imported into a non-operational "sandbox" environment on the target system. The service model remains hidden in the sandbox through the reconcile phase, and does not appear in the browser interface until you perform the final commit.

The import and reconcile process typically supports files that were exported by the same or different Resource Manager instance and version (older or newer), and Service Impact versions (older or newer). If an export file's content is incompatible with a version of import, incompatibilities can usually be resolved by adjusting the graphml file content or structure.

Preparing to import

Before you can import, you must copy the export file from your workstation to the target system master host and connect to that master host.

Prerequisite: Generate an export file (see [Exporting a service model](#) on page 34).

- 1 (*Resource Manager 5.x only*)
 - a Connect to the master host on which the `zenimpactstate` service is running.

```
ssh zenoss@impacthost.company.com
```

- b Create a directory for the exported file on the Control Center master host.

The directory that contains the exported file must be a local directory (not mounted) and must be readable, writable, and executable by all users. The following example create a directory in `/tmp`:

```
mkdir /tmp/impact && chmod 777 /tmp/impact
```

- c Change directory to the directory you created.
- For example:

```
cd /tmp/impact
```

- d Log in to the Control Center master host as a user with `serviced` CLI privileges.
 - e Copy the exported graphml file from your workstation to the new directory (if on the same host) or use `scp` to perform a network copy.

For example:

```
scp export_impact_svcname_20161215203455.graphml
zenossuser@impacthost.company.com
```

- f** Connect to the `zenimpactstate` container.

```
serviced service shell -i zenimpactstate
```

- g** In the new session, switch user from `root` to `zenoss`.

```
su - zenoss
```

- h** Change directory to the `pwd` directory:

For example:

```
cd /mnt/pwd
```

- 2** (*Resource Manager 4.2.x only*)

- a** From the originating Service Impact system, copy the exported file to a directory on the Resource Manager master host.
- b** Log in to the Resource Manager master host as `zenoss`.

Proceed with the initial import.

Initiating the import

In the initial import, service models from the exported file are imported into the sandbox, the product attempts to reconcile service model members from the originating system with entities on the target system, and output files are generated to show the result of the reconciliation attempt and actions to take.

Prerequisites:

- Generate an export file (see [Exporting a service model](#) on page 34).
- Copy the export file to the target system and log in (see [Preparing to import](#) on page 34).

- 1** Initiate the import.

```
zenimpactimport -i svc_export.graphml
```

The exported data is imported to the target system; the first reconciliation attempt is made, and generated text files report the results.

- 2** Review import results in file `svc_export.graphml.latest.txt`, and then proceed as follows:

- If **Nodes unreconciled** is 0, proceed with [Adding the service models to the target system](#) on page 39.
- If **Nodes unreconciled** is 1 or greater, proceed with [Reconciling originating and target system entities](#) on page 35.
- If the import ends in an error or you do not plan to commit the import, proceed with [Canceling the service model reconcile](#) on page 37.

Reconciling originating and target system entities

The reconcile phase is an iterative process in which you attempt to match devices or components from the originating system with their equivalents on the target system.

Multiple attempts to reconcile are often required before you can commit a service model that originated on another system. For faster progress, make multiple small changes and frequent attempts to reconcile.

Begin by reviewing the text files that the initial import generated. Files `svc_export.graphml.latest.txt` and `svc_export.graphml.0001.txt` are identical. Each attempt to reconcile generates a new "latest" file and a new, sequentially numbered file to provide a record of your changes. Do not modify the numbered files.

For each service model member, one of the following results or actions exists. When the attempt to reconcile succeeds, results are reported. When you must manually correct exceptions in the model, actions provide guidance. `<NodeID>` identifies the imported service model member. Do not modify the node ID. `<TargetNodeID>` is the DMD ID, name, or GUID of the target entity to apply to the imported service model member.

UNRECONCILED <NodeID>

The result for any member on the originating system that the service model references but which cannot be identified on the target system.

MAP <NodeID> <TargetNodeID>

The result if a matching entity was found on the target system or an action to take for an UNRECONCILED service model member. Replace UNRECONCILED with MAP and specify the DMD ID, name, or GUID for the target entity. For example, you might MAP to specify that device "stage-AppSvr1" should reference "prod-AppSvr1" in production.

```
MAP stage-AppSvr1 prod-AppSvr1
```

Note Some cases might require that you first define and model a device in Resource Manager so that the next reconcile attempt can detect it either automatically or manually through use of the MAP action.

CREATE <NodeID>

An action to take when no matching entity was found in the target system, but an entity can be created through import. For example, you might CREATE to add a device for production when a similar device does not exist in staging.

```
CREATE prod-AppSvr1
```

DELETE <NodeID>

The result when a matching entity was found in the target system, and it is marked to be deleted in the import file, or an action to take when an entity is not needed. Use DELETE to delete the device, not just remove it from the service model. Replace UNRECONCILED with DELETE and specify the DMD ID, name, or GUID.

For example, you might DELETE to remove a device in production.

```
DELETE prod-AppSvr1
```

If the `<NodeID>` does not exist in the target system, the action is converted to IGNORE.

IGNORE <NodeID>

An action to take for an imported service model member. For example, you might IGNORE because a similar entity does not exist in production.

```
IGNORE stage-AppSvr1
```

Before you commit, you must edit `svc_export.graphml.latest.txt` (or another version of the file) to specify action data. That is, replace UNRECONCILED entries with MAP, CREATE, DELETE, or IGNORE action syntax, and then attempt to reconcile.

Note In some cases, such as when target devices are not yet online, you might commit and make the service model active even with a few UNRECONCILED entries. Reconcile the remaining UNRECONCILED entries later by importing the same export file or a new file. This approach is useful if you plan to repeat the export to the same target, such as after upgrades or service model changes. You will not need to manually reconcile the same entries each time. You can also manually adjust committed service model definitions later in the target system by using the browser interface.

Attempting to reconcile

Prerequisites:

- Generate an export file (see [Exporting a service model](#) on page 34).
- Copy the export file to the target system and log in (see [Preparing to import](#) on page 34).
- Complete the initial import (see [Initiating the import](#) on page 35).
- Review [Reconciling originating and target system entities](#) on page 35.

For more information about the `zenimpactimport` command and options, see [zenimpactimport](#) on page 40 or access the help information:

```
zenimpactimport -h
```

- 1 In `svc_export.graphml.latest.txt` or another version of the file, replace UNRECONCILED with an action to specify how the exception should be reconciled.
- 2 Change directory to the directory that contains files `svc_export.graphml.latest.txt` and `svc_export.graphml.nnnn.txt`.
- 3 Initiate the reconcile attempt.

```
zenimpactimport -r svc_export.graphml
```

The `zenimpactimport` command attempts to reconcile the originating and target systems, and generates new versions of files `svc_export.graphml.latest.txt` and `svc_export.graphml.nnnn.txt`.

- 4 Review import results in file `svc_export.graphml.latest.txt`, and proceed as follows:
 - If **Nodes unreconciled** is 1 or greater, repeat the reconciliation process.
 - If **Nodes unreconciled** is 0, proceed with [Adding the service models to the target system](#) on page 39.
 - If the import ends in an error or you do not plan to commit the import, proceed with [Canceling the service model reconcile](#) on page 37.

Canceling the service model reconcile

You might need to remove an uncommitted import attempt from the target system sandbox and free up sandbox resources.

Prerequisites:

- Generate an export file (see [Exporting a service model](#) on page 34).
- Copy the export file to the target system and log in (see [Preparing to import](#) on page 34).
- Complete the initial import (see [Initiating the import](#) on page 35).

You can specify the import attempt to cancel in the following ways:

By import file name:

```
zenimpactimport --import filename.graphml --abort
```

By reconcile file name:

```
zenimpactimport --reconcileFile filename.graphml.latest.txt \  
--abort
```

By action file name:

```
zenimpactimport --actionFile filename.graphml*.latest.txt* \  
--abort
```

By source and version. First, generate a list of import attempts:

```
zenimpactimport --list-imports
```

Then specify source, version, or both:

```
zenimpactimport --source sourceID --abort
```

```
zenimpactimport --import-version versionID --abort
```

```
zenimpactimport --source sourceID \  
--import-version versionID --abort
```

The following example lists the import versions, the command to remove an uncommitted import, and the resulting list of import versions.

```
[zenoss@qa-set1-6-impact-zen graphml5-0-10]$  
zenimpactimport --list-imports  
Date                Source                Version                Status  
2018-02-13 14:59:36 d6fc-b4fe-00505694381d 1481662776493 COMMITTED  
2018-02-14 13:50:48 bd6d-a34a-0242ac110013 1481745048516 COMMITTED  
2018-02-14 14:22:11 bd70-945b-0242ac11003d 1481746931561 UNCOMMITTED  
2018-02-14 14:46:43 bd70-945b-0242ac11003d 1481748403174 UNCOMMITTED  
[zenoss@qa-set1-6-impact-zen graphml5-0-10]$  
zenimpactimport -r svc_export.graphml --import -version 1481746931561  
--abort  
INFO:zen.impact.import:Reading action data  
from svc_export.graphml.latest.txt  
INFO:zen.impact.import:Aborting import bd70-945b-0242ac11003d  
1481746931561 ...  
INFO:zen.impact.import:Printing report to svc_export.graphml.0003.txt  
INFO:zen.impact.import:Printing report to svc_export.graphml.latest.txt  
INFO:zen.impact.import:Nodes unreconciled: 0  
INFO:zen.impact.import:Import node status: aborted  
INFO:zen.impact.import:Done  
[zenoss@qa-set1-6-impact-zen graphml5-0-10]$  
zenimpactimport --list-imports  
Date                Source                Version                Status  
2018-02-13 14:59:36 d6fc-b4fe-00505694381d 1481662776493 COMMITTED  
2018-02-14 13:50:48 bd6d-a34a-0242ac110013 1481745048516 COMMITTED
```

```
2018-02-14 14:46:43 bd70-945b-0242ac11003d 1481748403174 UNCOMMITTED
```

Adding the service models to the target system

When the "latest" file identifies no UNRECONCILED nodes, commit the reconciled service models that are in the sandbox. After the commit, imported service models, organizers, logical nodes, global and contextual policies, and custom state providers are added to the target system. Service models are operationally active to process device and service events, and are listed in the **SERVICES** tab of the browser interface.

Prerequisites:

- Generate an export file (see [Exporting a service model](#) on page 34).
- Copy the export file and log in to the target system (see [Preparing to import](#) on page 34).
- Complete the initial import (see [Initiating the import](#) on page 35).
- Reconcile devices or components from the originating system with their equivalents in the target system (see [Attempting to reconcile](#) on page 37).
- Review the following considerations:
 - It is possible to commit the file with UNRECONCILED nodes, however, those members are lost from the resulting service model.
 - A final commit cannot be reversed. To change the model definition after committing, edit the service model definition in the **SERVICES** tab or delete the service model and start a new export process.

You can specify the import attempt to commit in the following ways:

By import file name:

```
zenimpactimport --import filename.graphml --commit
```

By reconcile file name:

```
zenimpactimport --reconcileFile filename.graphml.latest.txt \
--commit
```

By action file name:

```
zenimpactimport --actionFile filename.graphml*.latest.txt* \
--commit
```

By source and version. First, generate a list of import attempts:

```
zenimpactimport --list-imports
```

Then specify source, version, or both:

```
zenimpactimport --source sourceID --commit
```

```
zenimpactimport --import-version versionID --commit
```

```
zenimpactimport --source sourceID \
--import-version versionID --commit
```

Future export scenarios and machine learning

In the future, service models for dynamic services that you import might reference the same service model members that required manual reconcile actions in a previous import. The previous manual reconciliation and machine learning of mapped members simplifies each future export.

For example, the service model for dynamic service “S” in a Resource Manager staging environment refers to devices A,B, and C. In the production environment, the imported service model must be reconciled to map to the shared device A, and to equivalent devices X and Y instead of B and C.

Initial import of service model “S” into production

After the initial import, B and C were UNRECONCILED. You created MAP actions to reconcile B to X and C to Y, and then committed with `svc_export.graphml.latest.txt`.

Import 2 of service model “S” to the same production system

A few weeks later, an upgrade to service model “S” adds device “D.” The changed service model "S" with devices A,B, C, D is imported into production, where service model “S” must have entities A,X,Y, Z. Previously, you mapped and committed the service model with B mapped to X and C mapped to Y. After import 2, file `svc_export.graphml.latest.txt` shows only one UNRECONCILED entry, for added device “D.” Resource Manager learned and automatically mapped B and C to X and Y. This time, you only need to manually add a MAP action for “D” to “Z”, and then commit.

Import 3: initial import of the service model for dynamic service “T”

A third import occurs into production for a different service, “T.” Service model "T" contains devices A and D. After the import and automatic reconcile attempt, no UNRECONCILED entries exist for "T." That is, Resource Manager automatically found and mapped A, and used knowledge from your previous mapping of device D to Z for a different service import. In production, service model "T" contain devices A and Z.

Import 4: service model “T” with a different mapping of an entity than in "S" service context

A fourth import occurs, this time of service model “T” with service model members A, B, D. However, in the context of service “T,” device B should be mapped to device “P.” In production, service model “T” should contain service model members A, P, Z.

After the import and automatic reconcile attempt, no UNRECONCILED entries exist, but `svc_export.graphml.latest.txt` shows D mapped to Z and B mapped X. You must manually MAP B to P, reconcile, and then commit. As a result, the model for service "T" contains A, P, Z.

zenimpactimport

DESCRIPTION

`zenimpactimport` - Imports a *GraphML* file that contains Service Impact the service model definitions for a dynamic service; imports multiple subsequent calls to process new actions in order to reconcile a service model against the entities in Resource Manager.

The import process generates files that report the result of the import and recommend actions. The files follow the same format as the `--action` command line parameter. An action consists of an action keyword followed by parameters for that type of action. Actions and parameters are separated by white space, with each action on a separate line.

For more information about the import process, see *Importing service model definitions* on page 34.

SYNTAX

```
zenimpactimport (IMPORT_FILE | IMPORT_ID | ACTION_FILE) ACTIONS*
[TRANSACTION]
```

COMMANDS and OPTIONS

--import, -i

Perform the import operation.

--reconcile, -r

Perform the reconcile operation.

--help, -h

Show the help information.

--list-imports

Displays a list of existing imports and their status. Each import is uniquely identified by source and version identifiers.

--list-adapters

Displays a list of adapters that are installed to parse and recognize export file content and structure. The “zenoss” adapter expects a Zenoss Service Impact service export file in graphml format.

IMPORT_FILE

```
--import IMPORT_FILENAME [--adapter ADAPTER]
```

Performs the initial import of a file in graphml format that generates and attempt to reconcile service model members with entities on the target Resource Manager system.

IMPORT_FILENAME

The file that contains the data in a format that the ADAPTER recognizes. The default file name is `svc_export.graphml`.

ADAPTER

The name of an input adapter. Currently, only the default adapter for the Zenoss graphml export format is supported.

IMPORT_ID

```
--source SOURCEID --import-version VERSIONID
```

Performs the import of the attempt that you specify by source ID, version ID, or both. Each import is uniquely identified by source and version identifiers. To display imports, issue the `--list-imports` command.

The `*.latest.txt` file is rewritten by the latest command that uses the specified combination of *sourceID* and *versionID*.

ACTION_FILE

```
(--actionFile | --reconcileFile) FILENAME
```

Performs the import using data in the specified file.

FILENAME

A text file that contains reconcile actions. If no FILENAME is specified on the command line, file `svc_export.graphml.file.latest.txt` is used as input.

ACTIONS

```
--action ACTION
```

Provides a way to execute a single reconciliation action on the command line, rather than editing the input file or the parameter `--actionFile`. For more information about the actions, see [Reconciling originating and target system entities](#) on page 35.

CREATE *NodeID*

Creates the specified node on the target system.

DELETE *NodeID*

Deletes the specified node from the target system.

MAP *NodeID TargetNodeID*

Replaces the specified node on the originating system with the specified node on the target system.

IGNORE *NodeID*

Takes no action for the specified node.

UNRESOLVED *NodeID*

The specified node is referenced in the service model but it cannot be reconciled with a node on the target system.

TRANSACTION

```
{--abort|--commit}
```

Abort or commit the import attempt that is identified by its unique combination of source ID and version ID. Aborting destroys the import sandbox. Committing makes all entries in the input file permanent and operational in the target system. The last committed combination is the final result.

Example export file

The generated export file name is in the format

`export_impact_svcname_YYYYMMDDHHMMSS.graphml`, where

- `export` is the name of the action that is performed.
- `impact` is the name of the Resource Manager plugin that performs the action.
- `service` is the name of the service model, a service organizer, or dynamic services environment that you selected for export.
- `YYYYMMDDHHMMSS` is the timestamp of the export action.

For example, `export_impact_DynamicServices_20160331135219.graphml`.

Note This document refers to the file as `svc_export.graphml`.

The export file name is included in the names of the text files that the import process and each reconciliation attempt generate. For example, if file

`export_impact_DynamicServices_20160331135219.graphml` is imported, then the text file names are:

```
export_impact_DynamicServices_20160331135219.graphml.latest.txt
export_impact_DynamicServices_20160331135219.graphml.nnnn.txt
```

The following example shows the export file for a service named BizSvcA. The service model contains types DEVICE, SERVICE, and COMPONENT (PROP_element_type_id).

- The service is named BizSvcA (PROP_name) and it is a dynamic **service** (PROP_meta_type).
- The devices are named PROD_SQL_SERVER and vcloud-01.zenosslabs.com (PROP_name). They are a device and a vCloudCell, respectively (PROP_meta_type).
- The components are named APP_DB1 and tempdb (PROP_name). They are a WinSQLDatabases (PROP_meta_type).

- Four impact relationship edge members from each device and component.

Figure 8: Example export file

```
<?xml version="1.0" encoding="utf-8"?>
<graphml xmlns="http://graphml.graphdrawing.org/xmlns" xmlns:xsi="http://www.w3.org/2001/
XMLSchema-instance" xsi:schemaLocation="http://graphml.graphdrawing.org/xmlns http://
graphml.graphdrawing.org/xmlns/1.1/graphml.xsd">
  <key attr.name="PROP_production" attr.type="string" for="node" id="PROP_production"/>
  <key attr.name="INTRINSIC_STATE_AVAILABILITY" attr.type="string" for="node"
id="INTRINSIC_STATE_AVAILABILITY"/>
  <key attr.name="PROP_priority" attr.type="string" for="node" id="PROP_priority"/>
  <key attr.name="INTRINSIC_STATE_PERFORMANCE" attr.type="string" for="node"
id="INTRINSIC_STATE_PERFORMANCE"/>
  <key attr.name="RELATED_EVENT_IDS" attr.type="string" for="node" id="RELATED_EVENT_IDS"/>
  <key attr.name="NODE_TYPE" attr.type="string" for="node" id="NODE_TYPE"/>
  <key attr.name="DERIVED_STATE_PERFORMANCE" attr.type="string" for="node"
id="DERIVED_STATE_PERFORMANCE"/>
  <key attr.name="IN_ANY_CONTEXT" attr.type="string" for="node" id="IN_ANY_CONTEXT"/>
  <key attr.name="PROP_element_type_id" attr.type="string" for="node"
id="PROP_element_type_id"/>
  <key attr.name="DERIVED_STATE_AVAILABILITY" attr.type="string" for="node"
id="DERIVED_STATE_AVAILABILITY"/>
  <key attr.name="ID" attr.type="string" for="node" id="ID"/>
  <key attr.name="PROP_name" attr.type="string" for="node" id="PROP_name"/>
  <key attr.name="INTRINSIC_STATE_CAPACITY" attr.type="string" for="node"
id="INTRINSIC_STATE_CAPACITY"/>
  <key attr.name="PROP_meta_type" attr.type="string" for="node" id="PROP_meta_type"/>
  <key attr.name="ORGANIZER" attr.type="string" for="node" id="ORGANIZER"/>
  <key attr.name="DESCRIPTION" attr.type="string" for="node" id="DESCRIPTION"/>
  <key attr.name="LN_CRITERIA" attr.type="string" for="node" id="LN_CRITERIA"/>
  <key attr.name="LN_AVAILABILITY_MAP" attr.type="string" for="node"
id="LN_AVAILABILITY_MAP"/>
  <key attr.name="LN_PERFORMANCE_MAP" attr.type="string" for="node"
id="LN_PERFORMANCE_MAP"/>
  <key attr.name="CUSTOM_STATE_PROVIDER" attr.type="string" for="node"
id="CUSTOM_STATE_PROVIDER"/>
  <key attr.name="LABEL" attr.type="string" for="edge" id="LABEL"/>
  <graph edgedefault="directed" id="d96ce0f4-2d06-11e6-b939-0242ac110026">
    <node id="8ddcc69b-e6a7-4370-8f3d-4a53f897b9f4">
      <data key="PROP_production">1000</data>
      <data key="PROP_priority">3</data>
      <data key="NODE_TYPE">ELEMENT</data>
      <data key="DERIVED_STATE_PERFORMANCE">__DERIVED_STATE__|ACCEPTABLE|</data>
      <data key="IN_ANY_CONTEXT">true</data>
      <data key="PROP_element_type_id">DEVICE</data>
      <data key="DERIVED_STATE_AVAILABILITY">__DERIVED_STATE__|DOWN|</data>
      <data key="ID">8ddcc69b-e6a7-4370-8f3d-4a53f897b9f4</data>
      <data key="PROP_name">PROD_SQL_Server</data>
      <data key="PROP_meta_type">Device</data>
      <data key="CUSTOM_STATE_PROVIDER">{}</data>
    </node>
    <node id="fc6ab26d-75cd-491a-8c71-75fb25845f97">
      <data key="PROP_production">1000</data>
      <data key="NODE_TYPE">ELEMENT</data>
      <data key="DERIVED_STATE_PERFORMANCE">__DERIVED_STATE__|ACCEPTABLE|</data>
      <data key="IN_ANY_CONTEXT">true</data>
      <data key="PROP_element_type_id">COMPONENT</data>
      <data key="DERIVED_STATE_AVAILABILITY">__DERIVED_STATE__|UP|</data>
      <data key="ID">fc6ab26d-75cd-491a-8c71-75fb25845f97</data>
      <data key="PROP_name">APP_DB1</data>
      <data key="PROP_meta_type">WinSQLDatabase</data>
      <data key="CUSTOM_STATE_PROVIDER">{}</data>
    </node>
    <node id="d66416a5-7db7-47df-b9ab-ed9422700efe">
      <data key="PROP_production">1000</data>
      <data key="NODE_TYPE">ELEMENT</data>
      <data key="DERIVED_STATE_PERFORMANCE">__DERIVED_STATE__|ACCEPTABLE|</data>
      <data key="IN_ANY_CONTEXT">true</data>
      <data key="PROP_element_type_id">COMPONENT</data>
      <data key="DERIVED_STATE_AVAILABILITY">__DERIVED_STATE__|UP|</data>
      <data key="ID">d66416a5-7db7-47df-b9ab-ed9422700efe</data>
      <data key="PROP_name">tempdb</data>
      <data key="PROP_meta_type">WinSQLDatabase</data>
      <data key="CUSTOM_STATE_PROVIDER">{}</data>
    </node>
  </graph>
</graphml>
```

```

</node>
<node id="0dcda951-9885-449c-af1b-0adb37aeec95">
  <data key="IN_ANY_CONTEXT">true</data>
  <data key="PROP_element_type_id">SERVICE</data>
  <data key="DERIVED_STATE_AVAILABILITY">__DERIVED_STATE__|DOWN|</data>
  <data key="ID">0dcda951-9885-449c-af1b-0adb37aeec95</data>
  <data key="NODE_TYPE">SERVICE</data>
  <data key="PROP_name">BizSvcA</data>
  <data key="DERIVED_STATE_PERFORMANCE">__DERIVED_STATE__|ACCEPTABLE|</data>
  <data key="PROP_meta_type">DynamicService</data>
  <data key="ORGANIZER">/Zreconsvcs</data>
  <data key="DESCRIPTION"/>
</node>
<node id="6fca9724-6c92-4fd9-b8fa-c72bb133fda8">
  <data key="PROP_production">1000</data>
  <data key="PROP_priority">3</data>
  <data key="NODE_TYPE">ELEMENT</data>
  <data key="DERIVED_STATE_PERFORMANCE">__DERIVED_STATE__|ACCEPTABLE|</data>
  <data key="IN_ANY_CONTEXT">true</data>
  <data key="PROP_element_type_id">DEVICE</data>
  <data key="DERIVED_STATE_AVAILABILITY">__DERIVED_STATE__|UP|</data>
  <data key="ID">6fca9724-6c92-4fd9-b8fa-c72bb133fda8</data>
  <data key="PROP_name">vcloud-01.zenoss.com</data>
  <data key="PROP_meta_type">vCloudCell</data>
  <data key="CUSTOM_STATE_PROVIDER">{}</data>
</node>
<edge id="2" source="8ddcc69b-e6a7-4370-8f3d-4a53f897b9f4" target="0dcda951-9885-449c-af1b-0adb37aeec95">
  <data key="LABEL">IMPACTS</data>
</edge>
<edge id="0" source="fc6ab26d-75cd-491a-8c71-75fb25845f97" target="0dcda951-9885-449c-af1b-0adb37aeec95">
  <data key="LABEL">IMPACTS</data>
</edge>
<edge id="1" source="d66416a5-7db7-47df-b9ab-ed9422700efe" target="0dcda951-9885-449c-af1b-0adb37aeec95">
  <data key="LABEL">IMPACTS</data>
</edge>
<edge id="3" source="6fca9724-6c92-4fd9-b8fa-c72bb133fda8" target="0dcda951-9885-449c-af1b-0adb37aeec95">
  <data key="LABEL">IMPACTS</data>
</edge>
</graph>
</graphml>

```

Example import results file

The initial import generates `svc_export.graphml.latest.txt` to report the results of the process. The following example shows unreconciled, created, and mapped service model members.

Be aware that the `svc_export.graphml.latest.txt` is rewritten by the latest command that uses the specified combination of *sourceID* and *versionID*.

UNRECONCILED

In this example, the initial import could not match databases DB3From, DB2From, and tempdbFrom with databases on the target system. Manual reconciliation is needed. For UNRECONCILED entries, known information is displayed so that you can search the target system for a corresponding entity. In the example, `PROP_meta_type` indicates that a WinSQLDatabase is referenced, `PROP_element_type_id` shows that it is a COMPONENT, and `PROP_name` shows the database name. After the EXTERNAL_ID has been reconciled, subsequent outputs do not include these details.

CREATE

For imported service DBSvcExpTarget, a matching service does not exist on the target system. The import command generated a CREATE statement for that service and displays related details about the service that is being created.

MAP

For the SQL Server device and database from the originating system, the import process found and mapped the same members on the target system. Details about the members are not displayed because manual reconciliation is not needed.

```
[zenoss@fde4efeccf58 pwd]$ cat DBSvcExpFROM.graphml.latest.txt
SOURCE      d96ce0f4-2d06-11e6-b939-0242ac110026
VERSION     1481917173580 # 2016-12-16 19:39:33

# CUSTOM_STATE_PROVIDER:      {}
# DERIVED_STATE_AVAILABILITY:  __DERIVED_STATE__|UP|
# DERIVED_STATE_PERFORMANCE:  __DERIVED_STATE__|ACCEPTABLE|
# EXTERNAL_ID:                d05e3670-0bbf-4e76-bcd2-a8a1cd333333
# ID:                          d05e3670-0bbf-4e76-bcd2-a8a1cd333333
# IN_ANY_CONTEXT:             true
# NODE_TYPE:                  ELEMENT
# PROP_element_type_id:       COMPONENT
# PROP_meta_type:             WinSQLDatabase
# PROP_name:                  DB3From
# PROP_production:            1000
UNRECONCILED    d05e3670-0bbf-4e76-bcd2-a8a1cd333333

# CUSTOM_STATE_PROVIDER:      {}
# DERIVED_STATE_AVAILABILITY:  __DERIVED_STATE__|UP|
# DERIVED_STATE_PERFORMANCE:  __DERIVED_STATE__|ACCEPTABLE|
# EXTERNAL_ID:                d4176972-bb0e-44b3-90d2-bb8e96222222
# ID:                          d4176972-bb0e-44b3-90d2-bb8e96222222
# IN_ANY_CONTEXT:             true
# NODE_TYPE:                  ELEMENT
# PROP_element_type_id:       COMPONENT
# PROP_meta_type:             WinSQLDatabase
# PROP_name:                  DB2From
# PROP_production:            1000
UNRECONCILED    d4176972-bb0e-44b3-90d2-bb8e96222222

# CUSTOM_STATE_PROVIDER:      {}
# DERIVED_STATE_AVAILABILITY:  __DERIVED_STATE__|UP|
# DERIVED_STATE_PERFORMANCE:  __DERIVED_STATE__|ACCEPTABLE|
# EXTERNAL_ID:                d66416a5-7db7-47df-b9ab-ed9422666666
# ID:                          d66416a5-7db7-47df-b9ab-ed9422666666
# IN_ANY_CONTEXT:             true
# NODE_TYPE:                  ELEMENT
# PROP_element_type_id:       COMPONENT
# PROP_meta_type:             WinSQLDatabase
# PROP_name:                  tempdbFrom
# PROP_production:            1000
UNRECONCILED    d66416a5-7db7-47df-b9ab-ed9422666666

# DERIVED_STATE_AVAILABILITY:  __DERIVED_STATE__|DOWN|
# DERIVED_STATE_PERFORMANCE:  __DERIVED_STATE__|ACCEPTABLE|
# DESCRIPTION:
# EXTERNAL_ID:                1f3e3f0b-35a4-4851-8863-b94f36AAAAAA
# ID:                          1f3e3f0b-35a4-4851-8863-b94f36AAAAAA
# IN_ANY_CONTEXT:             true
# NODE_TYPE:                  SERVICE
# ORGANIZER:                  /Zreconsvcs
# PROP_element_type_id:       SERVICE
# PROP_meta_type:             DynamicService
# PROP_name:                  DBSvcExpTarget
```

```
# RECONCILE_ACTION:          CREATE
CREATE          1f3e3f0b-35a4-4851-8863-b94f36AAAAAA

MAP          8ddcc69b-e6a7-4370-8f3d-4a53f897b9f4 /zport/dmd/Devices/Server/
Microsoft/Windows/devices/10.111.5.81

MAP          fc6ab26d-75cd-491a-8c71-75fb25845f97 /zport/dmd/Devices/Server/
Microsoft/Windows/devices/10.111.5.81/
                                                    os/winsqlinstances/
INSTANCE1/databases/INSTANCE15
```

4

Browser interface resources

Service Impact resources are available on the **SERVICES** tab of the Resource Manager browser interface.

Adding an Impact Services portlet

You can customize the Resource Manager dashboard by adding an Impact Services portlet.

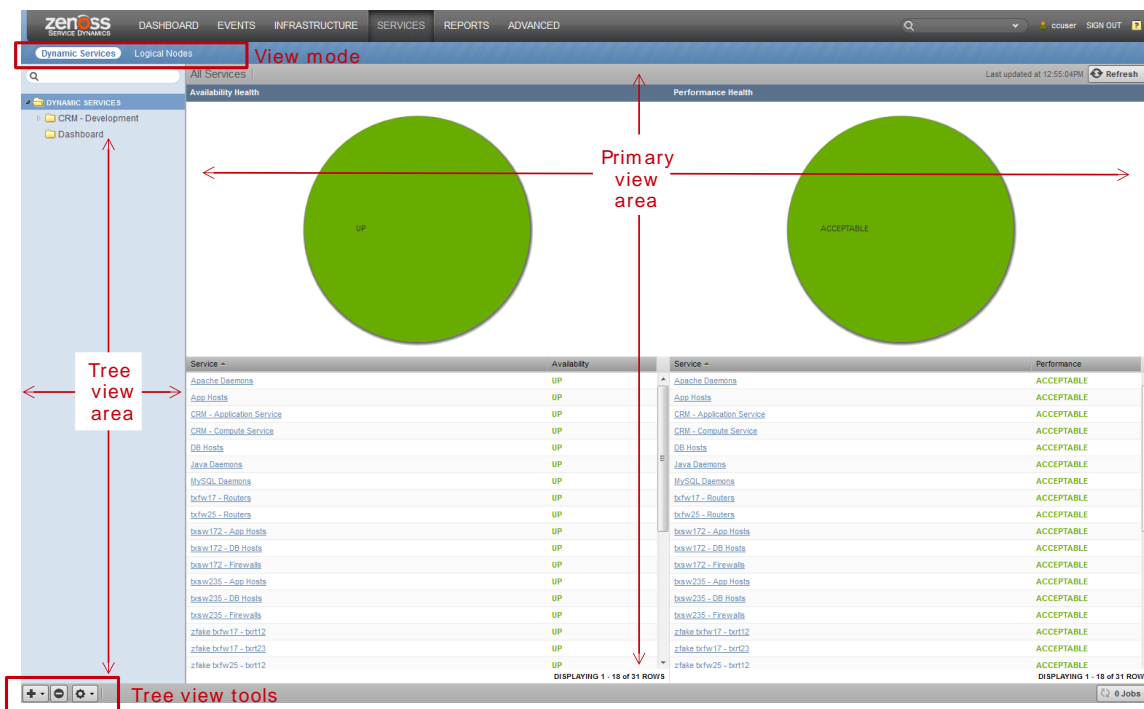
Prerequisites: Service Impact must be installed and at least one dynamic service must exist.

- 1 In the Resource Manager browser interface, choose **DASHBOARD**.
- 2 In the top-right of the main dashboard area, click **Add**, and then choose **Add portlet**.
- 3 From the **Portlet** drop-down list, choose **Impact Services**.
- 4 In the **Preview** pane, choose the services that you want to show in the portlet, and then click **Add**.

Service Impact home page

The Service Impact feature home page in the Resource Manager browser interface displays the health summaries of all services.

From the home page, you can view information in **Dynamic Services** mode or **Logical Nodes** mode. In both view modes, the Service Impact home page includes the tree view area and its tools, and the primary view area.



Tree view area

The tree view area displays service nodes and logical nodes in alphabetical order. You can create organizers and order them as you wish. To move service nodes and logical nodes into organizers, drag them into the tree view.

Tool	Menu	
	Dynamic Services view mode	Logical Nodes view mode
	Add Dynamic Service Add Dynamic Service Organizer	Add Logical Node Add Logical Node Organizer
	(no menu)	(no menu)
	View and Edit Details Clone Service... Export Selected	(no menu)

Primary view area

The content of the primary view area depends on the view mode and the item that you select in the tree view area.

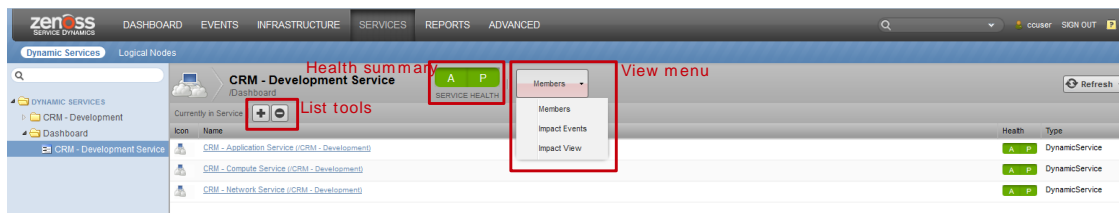
View mode	Tree view selection	Primary view contents
Dynamic Services	DYNAMIC SERVICES (the root organizer)	The availability and performance health summaries of all services.

View mode	Tree view selection	Primary view contents
	An organizer	The availability and performance health summaries of the services that the organizer contains.
	A dynamic service (service model)	The Members of the selected service. From this view, select Impact Events on page 50 or Impact View on page 51.
Logical Nodes	<ul style="list-style-type: none"> ■ LOGICAL NODES (the root organizer) ■ An organizer 	A blank logical node details view. (See Logical node details view on page 57.)
	A logical node	The details view of the selected logical node. (See Logical node details view on page 57.) No other views are associated with logical nodes.

Members view



The **Members** view provides details about a service member, including the list of its associated members.

The following example shows the **Members** view of a service member, with key features highlighted, and the view menu selected.



List tools

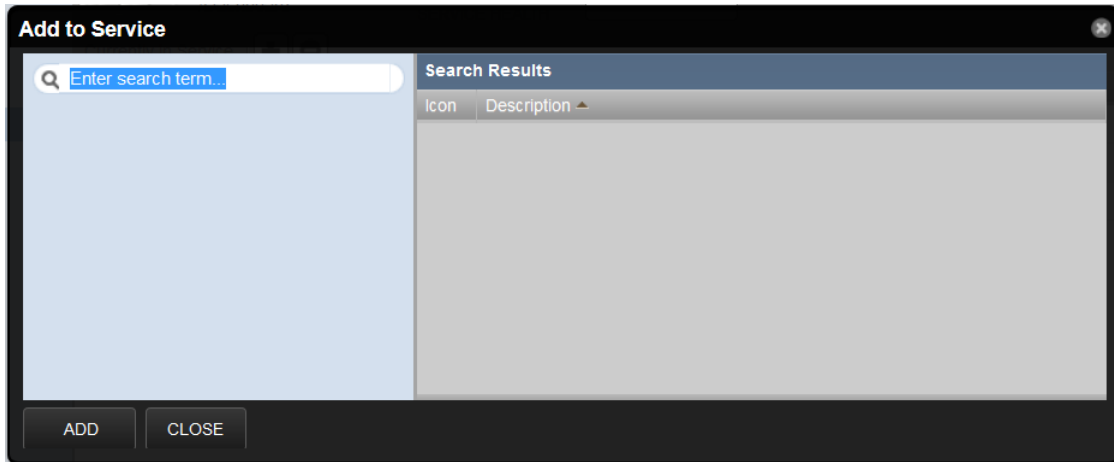
The list tools add or remove members from a service model.

Tool	Function
	Display the Add to Service dialog box.
	Remove a selected member from the service model.

Add to Service dialog box

Use the **Add to Service** dialog box to find and add members to a service model.

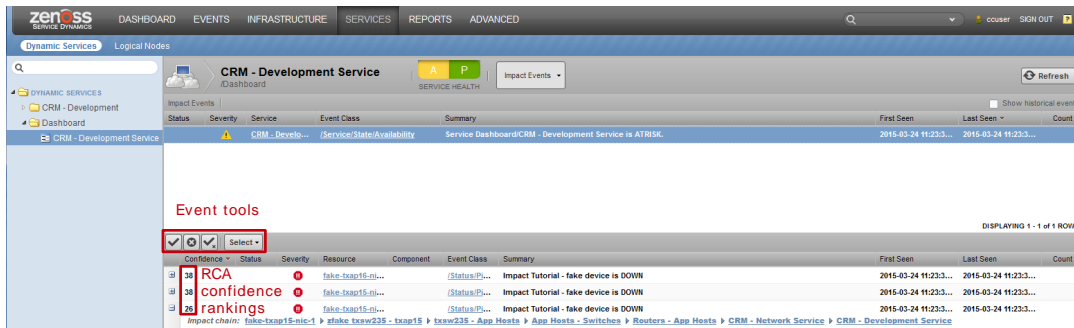
In the search field, enter the string to find. You can search for a specific device or component name, or component type (such as interfaces and OS processes) or group type (such as location and system). As you type characters, Service Impact displays matches.



Impact Events

The **Impact Events** view shows summary and detail information about events that are affecting a service.


The following example shows an **Impact Events** view with key features highlighted. The last event in the details area is expanded to show the **Impact chain**, which is the hierarchy of service model members that are associated with the event.



Event tools

The event tools provide options for manipulating the list of events.

Tool	Function
	Acknowledge the selected event.
	Close the selected event. The event is moved to the archive.
	Undo acknowledgement of the selected event.

Tool	Function
	Display the event selection menu.
Select All	Select all events in the list.
None	Deselect all selected events.

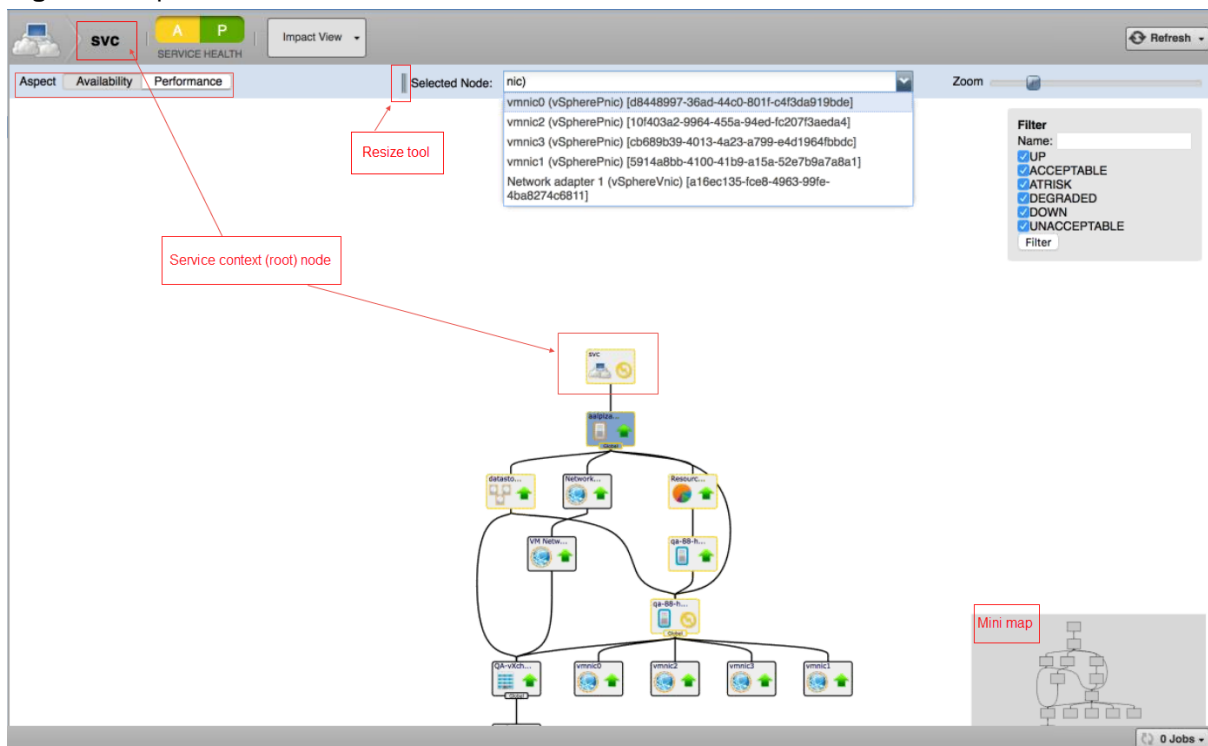
Root-cause analysis (RCA) confidence rankings

The second column of the details list contains the event's confidence ranking. Service Impact knows which members affect which service models, and automatically performs RCA when an event occurs. The analysis yields a probability value that an event is the cause of the service's current state. Often, multiple events contribute to a state, so the confidence rankings enable you to quickly focus resources on the most likely cause.

Impact View

The **Impact View** displays the interactive Service Impact graph of a service model and provides several views and tools.

Figure 9: Impact View with views and tools identified



Service context (root) node name

In the upper left above the **Aspect** views, the service context (root) node of the graph is displayed. Click the name to make the root node the selected node and display its name in the **Selected node** field.

Aspect views: Availability, Performance

Use aspect views to display and filter on availability or performance states in the graph.

Selected Node

Displays the name and type of the node that you have chosen, or the service context (root) node if you have not chosen a node. For example, DB Hosts (Organizing Group) is the node named DB Hosts, which is an organizing group type. When you first access the **Impact View**, the **Selected Node** field displays the root node. To resize the **Selected Node** field, hover the pointer to the left of the field label to display the resize tool, and then drag the tool left or right. When you choose a node from the list or click a tile in the graph, that node appears in **Selected Node** and the node tile is centered in the graph.

To search for specific nodes, begin typing a string in the **Selected Node** field. The field's drop-down list displays matching nodes in the following format:

```
node_name (node_type) [UID]
```

Tips for searching nodes

- Strings are not case sensitive.
- Regular expressions are not supported.
- You can search for nodes by name, type, or system-generated unique identifier (UID).
- To search for nodes by name only, suffix the search string with an open parenthesis. For example, the following string matches only node names that end with "Hosts," such as the DB Hosts node.

```
Hosts (
```

However, the following string matches nodes with "hosts" in the name or type.

```
hosts
```

- To search for nodes by type only, prefix the search string with an open parenthesis. For example, the following string matches only node types beginning with "vSphere," such as the vSpherePnic type.

```
(vSphere
```

However, the following string matches nodes with "vsphere" in the name or type.

```
vSphere
```

- To search for nodes that contain the same character string in the UID, prefix the search string with an open bracket. For example, the following string matches only nodes with a UID beginning with "G7M," such as G7M589.

```
[G7M
```

However, the following string matches nodes with "G7M" in the name, type, or UID.

```
G7M
```

Zoom

To increase or decrease the size of tiles in the graph, use the mouse wheel or drag the **Zoom** control in the upper right corner of the view.

Node tools

To see a summary of a node in a Service Impact graph, hover the pointer over its tile. Summary information includes the node name, availability state, performance status, production status, and type.

To display the following menu options for a node, position the pointer over it and right-click:

Toggle Children

Hide or display the node's child and descendent nodes.

Edit Impact Policies

Display the **Impact Policies** dialog box. See [Impact Policies dialog box](#) on page 54.

Graph and filter tools (right-click menu)

To display the following **Impact View** menu options, position the pointer in the primary view area and right-click.

Fit Graph to Window

Adjust the size of the graph so that its dimensions match the display area.

Show All

Display all nodes in the graph, including those that are automatically added by ZenPacks.

Collapse All

Display only the top-level node of the graph.

Compact View

Display only the service model members that you have manually added and the immediate children of a service or service group.

Toggle Rainbows

Display colored tabs at the right edge of the node, indicating the number of critical, error, and warning events associated with the node.

Toggle Filter

Hide or display the filter controls.

Use filter check boxes to select availability and performance states by which to filter. By default, all states are selected for filtering. To restrict displayed nodes to specific states, such as DOWN and UNACCEPTABLE, select only those check boxes. (Changes to the selected states apply to all dynamic services because state is a global feature.)

To filter by complete or partial node name, use the **Name** field. To perform the filter click **Filter**.

Export Graph Image

Creates a file of the graph image with the default name graph.png.

Enable center view**Disable center view**

Shift the graph so that the selected node is in the center. The zoom level does not change. You can zoom in and out on the node in the center of the graph. Choose this option to recenter a specific node.

Jump to Selected



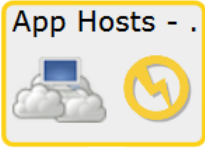
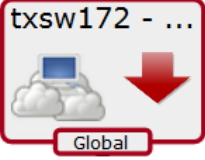
Shift the graph so that the node that is shown in the **Selected Node** field is in the center.

Jump to Context (root)

Shift the graph so that the service context (root) node of the graph is in the center.

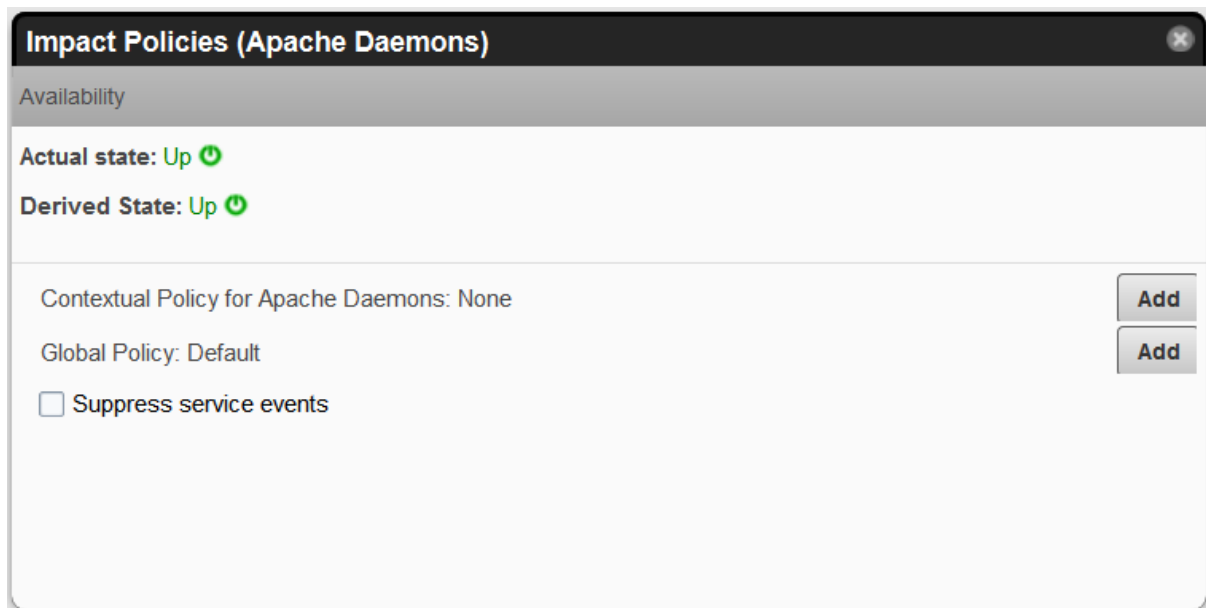
Node tile

The node tile graphically represents a server or service node and information about it, as described in the following table:

Node tile example	Description
 <p>The tile shows the name 'epowell-de...' at the top. Below the name is a computer icon with a penguin on it and a green arrow pointing up. The entire tile is enclosed in a black border.</p>	<p>This tile represents a Linux server named <code>epowell-debug.zenoss.loc</code>. (The name is shortened to fit inside the tile.)</p> <ul style="list-style-type: none"> ■ The computer icon and penguin image represent the node's type, Linux server. ■ The green arrow and the black border represent the node's availability state, UP.
 <p>The first tile, 'Firewalls - ...', has a computer icon, a green arrow pointing up, and a 'Global' label at the bottom. It has three colored tabs (red, yellow, green) on the right side. Below the tile are three blue dots. The second tile, 'interna...', has a computer icon, a red arrow pointing down, and a '1' in a red circle on the right side. It has a red dashed border.</p>	<p>This tile represents a service node with a global policy. Event rainbows are displayed.</p> <ul style="list-style-type: none"> ■ The event rainbow is the colored tabs on the right side of the tile. From top to bottom, the rainbow shows the counts of critical, error, and warning events that are associated with the node. The first example node has no events, so no numbers are displayed. The second example node has 1 critical error. ■ When you click an event rainbow tab, Resource Manager displays the overview page of the node. ■ This tile includes three dots immediately below its bottom border. The dots represent descendant nodes. To display the descendant nodes, double-click the node tile.
 <p>The tile shows the name 'App Hosts - .' at the top. Below the name is a computer icon and a yellow lightning bolt icon. The entire tile is enclosed in a yellow border.</p>	<p>This tile represents a service node. The yellow icon and border represent the node's availability state, ATRISK.</p>
 <p>The tile shows the name 'txsw172 - ...' at the top. Below the name is a computer icon and a red arrow pointing down. A 'Global' label is at the bottom. The entire tile is enclosed in a red border.</p>	<p>This tile represents a service node with a global policy. The red arrow and the red border of the tile represent the node's availability state, DOWN.</p>

Impact Policies dialog box

In the **Impact Policies** dialog box, add or edit state triggers for contextual or global policies.



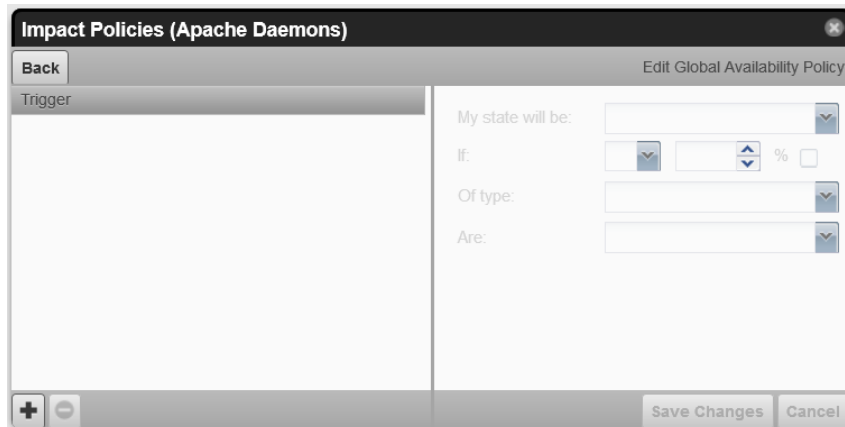
To prevent sending service events when changes affect a node, check **Suppress service events** (not shown). Choose this option when a service node is used solely to group child nodes.

If a custom state provider is not associated with a node, the dialog box does not contain the option to edit the custom state provider.

For a policy or custom state provider, click **Add** or **Edit** to access options.

Edit Policy options

Use the **Edit Policy** options to add or edit state triggers for contextual or global policies.



To add a trigger, in the lower-left corner click Add. On the right side of the dialog box, modify the trigger fields (described in the following table), and then click **Save Changes**.

Field	Description
My state will be	The new state, if the trigger applies. Possible states: DOWN, DEGRADED, ATRISK, UP.
If	The conditions that trigger a state change. To specify a percentage rather than an absolute value, click %.

Field	Description
Of type	Restrict the type of child node to which the trigger applies. Types other than Any are exclusive.
Are	The state of child nodes that cause an evaluation of this trigger. Possible states: DOWN, DEGRADED, ATRISK, UP.

Edit Custom State Provider options

Use a custom state provider to add state triggers (rules) for custom device and component service model members. You can customize Resource Manager to gather state data from events that belong to other classes. For example, customize state providers to define state triggers for members that are monitored through customized classes that the ZenVMware and CiscoUCS ZenPacks provide.

Field	Description
Event Class	Choose the Resource Manager event class to monitor. You can configure one event class per device.
Event severity fields (Critical, Error, Warning, Info, Debug, Clear)	Choose the state for this member if the event severity is observed. Possible states are UP, ATRISK, DEGRADED, DOWN.
Apply to	Choose the nodes (components) to which the state override applies. <ul style="list-style-type: none"> ■ This node only: The selected component on the specific device. ■ Nodes of the same type on the same device: The same component type on the same device. If a component is associated with another device, it is not affected in that context. ■ Nodes of the same type in the same device class: The same component type on any device with the same device class. ■ Nodes of the same type system-wide: The same component type, regardless of the device or the device class.

Logical node details view

The logical node details view enables you to create and edit customizable members that capture specific Resource Manager event states.

Logical nodes allow you to represent multiple resources or services; they rarely relate to events from a single member. For resources that Resource Manager monitors, logical nodes enable you to capture event states from arbitrary classes. Logical nodes might be external events from third-party monitoring products that are not monitored by Resource Manager.

For example, a bookstore website uses a third-party financial service to process credit cards. Resource Manager has no access to that third-party environment, but the monitoring systems forward events to Resource Manager. Their member names are prefixed with FincCC (that is, FincCC-AppSvr01, FincCC-AppSvr02, FincCC-OraDB01, and so on). To match events from that system and relate them to the "bookstore" service, you create a logical node that matches any event with a resource name that includes the prefix FincCC.

A logical node can be a child of a service model member, but no other type.

The screenshot shows a web form for configuring a logical node. The fields are as follows:

- Name:** zenoss.com
- Description:** (empty)
- Criteria:** all of the following rules: Summary contains fakeInternet
- Availability State:**
 - Events for this node in this event class: /Status/Ping
 - will result in these availability states:
 - Critical: DOWN
 - Error: DOWN
 - Warning: ATRISK
 - Info: (empty)
 - Debug: (empty)
 - Clear: UP
- Performance State:**
 - Events for this node in this event class: (empty)
 - will result in these performance states:
 - Critical: (empty)

Buttons: Save, Cancel

Criteria

The **Criteria** rules allow you to define event triggers and associate the triggers with availability and performance states.

Availability state

In **Events for this node in this event class**, specify the event class or subclass that is associated with the trigger.

The following examples illustrate how to specify event classes.

/Status

Only the /Status class.

/Status/Web

Only the Web subclass of /Status

/Status/

The /Status class and all of its subclasses.

In **will result in these availability states**, map event severity levels to availability states.

Performance State

The fields and options in this area differ only in that the mapping of event severity levels to performance states uses different states.

5

Configuring Service Impact

This chapter describes the Service Impact configuration options.

Production state propagation threshold

Service Impact relies on device production states to decide whether to propagate availability and performance states within a service graph.

The following table characterizes the device production states that are available in Resource Manager.

Production state	Devices monitored?	Appear on dashboard?
Production	Yes	Yes
Pre-Production	Yes	No
Test	Yes	No
Maintenance	Yes	Might appear
Decommissioned	No	No

Devices are displayed in the *Impact View* as determined by production state and production state propagation threshold. Each threshold setting indicates which states are considered to be "in production." For example:

- The threshold value Production indicates that only production-level devices are to be considered "in production."
- The threshold value Decommissioned indicates that devices in all production states are to be considered "in production."

In the **Impact View**, background color indicates the node's production state. In the following table,

- FALSE indicates that the node is displayed with a gray background, and its state is not propagated upwards to other members.
- TRUE indicates that the node is displayed with a white background, and interacts with other members as "in production."

Device production state	Threshold= Production	Threshold= Pre-Production	Threshold= Test	Threshold= Maintenance	Threshold= Decommissioned
Decommissioned	FALSE	FALSE	FALSE	FALSE	TRUE

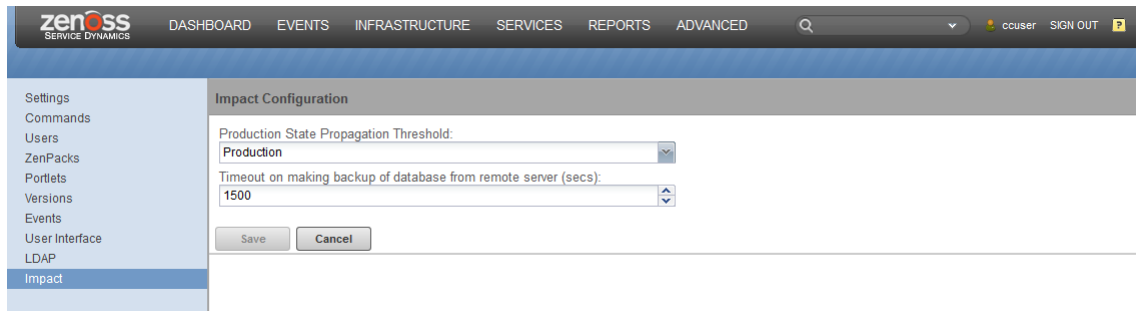
Device production state	Threshold= Production	Threshold= Pre-Production	Threshold= Test	Threshold= Maintenance	Threshold= Decommissioned
Maintenance	FALSE	FALSE	FALSE	TRUE	TRUE
Test	FALSE	FALSE	TRUE	TRUE	TRUE
Pre-Production	FALSE	TRUE	TRUE	TRUE	TRUE
Production	TRUE	TRUE	TRUE	TRUE	TRUE

For more information about production states, refer to the *Zenoss Resource Manager Administration Guide*.

Configuring the production state propagation threshold

The default value of the Service Impact production state propagation threshold is Production. You can select a different production state for the threshold. Zenoss recommends changing this threshold only during installation.

- 1 Log in to the Resource Manager browser interface as a user with ZenManager or Manager privileges.
- 2 Select the **ADVANCED** tab, and then click the **Impact** link in the left column.



- 3 From the **Production State Propagation Threshold** list, select a new production state, and then click **Save**. Devices are evaluated against the new threshold at the next modelling interval. To update all devices before the next modelling interval, perform a graph update.

Service Impact server configuration files

The `$IMPACT_HOME/etc` directory contains Service Impact server configuration files.

`$IMPACT_HOME/etc/zenoss-dsa-amqpconf.properties`

This file contains properties for the Service Impact host's connection to the RabbitMQ server.

`$IMPACT_HOME/etc/zenoss-dsa.env`

This file contains the `JVM_ARGS` variable definition.

`$IMPACT_HOME/etc/zenoss-dsa.properties`

This file contains general Service Impact configuration properties, including properties for backups, log file management, and remote debugging.

Note Configuration settings and their associated default values might appear commented out in the file; however, they are automatically activated at installation. To modify the default value, you must first uncomment the setting. Before you change values in any file in this directory, consult with Zenoss Support.

Configuration file best practices

Zenoss recommends that you edit configurations through the Control Center browser interface. Using this method, edits are preserved and pushed to the Service Impact container when restarted. When Service Impact restarts, a new container is created. Any changes applied using the command are lost when Service Impact is restarted.

Use the Control Center browser interface to edit and preserve changes to `/opt/zenoss_impact/etc/logback.xml`. Modifying `logback.xml` in the Control Center browser interface requires a restart of the Service Impact service for the logging changes to take effect. If you use the command line to implement logging changes, they are implemented immediately; however, the changes made using the command line are lost when Service Impact is restarted.

The following sections describe how to edit Service Impact configuration files for Resource Manager 5.x and 4.2.x.

Editing server configuration files with Control Center (Resource Manager 5.2.x and later)

To test changes in a pre-production environment, modify configuration file settings directly in the container. When Service Impact is restarted, all settings are returned to their default value.

This procedure describes how to use Control Center to edit Service Impact server configuration files with Resource Manager 5.2.x and later.

- 1 Log in to the Control Center master host as `root`, or as a user with superuser privileges.
- 2 In the **Applications** table, click **Zenoss.resmgr**.
- 3 Scroll down to the **Services** list and select **Impact**.
- 4 Under **Configuration Files**, click **Edit** to the right of `/opt/zenoss/etc/dsa.zenoss-dsa.properties`.

The `zenoss-dsa.properties` file is displayed in the **Edit Configuration** window.

Note Modifying properties might change the performance of Service Impact. Before you modify default values, consult Zenoss Support.

- 5 Make changes and then click **Save**.
- 6 To activate the changes, restart Service Impact.

Editing server configuration files in the Impact container (Resource Manager 4.2.5)

- 1 Log in to the Impact container as the `zenossimpact` user.
 - a Log in to the Control Center master host as a user with `serviced` CLI privileges.
 - b Start an interactive session in the Impact container.

```
serviced service attach impact
```

- c In the new session, switch user to `zenossimpact`.

```
su - zenossimpact
```

- 2 Log in to the Service Impact server host as `zenossimpact`.
- 3 Change directory to the configuration properties directory.

```
cd $IMPACT_HOME/etc
```

- 4 Edit configuration files.
- 5 Restart Service Impact, and then log off the Service Impact server host.
 - a Restart the Service Impact.

```
service zenoss_impact restart
```

- b Log off.

```
exit
```

Targeted graph update

Occasionally, Service Impact receives information about an unknown component or device that is needed for an event. When this occurs, a targeted graph update is performed.

To rapidly update its knowledge of an unknown entity and its impact model, Service Impact targets modeling of only that entity's subgraphs. To allow the modeling workflow to complete, a "placeholder" is immediately created for the unknown entity. Impact graphs are quickly refreshed without delay to event processing.

The targeted graph update allows mission-critical event and service event monitoring and alerting to resume with minimal downtime. During the restore and rebuild of the Neo4j graph database, service event processing resumes quickly, regardless of the size and number of service models.

You can tune targeted graph updates by changing configuration parameters.

Targeted graph update occurs automatically and requires no user intervention. However, in unusual circumstances, Zenoss Support might recommend the following actions:

- Adjusting configuration parameters to better tune performance.
- Repairing information by manually rebuilding impact models with missing entities and impact relationships.

Tuning targeted graph update

Adjusting targeted graph update settings might cause performance degradation. Change settings only if instructed to do so by Zenoss Support.

- 1 To change settings, open the `/opt/zenoss_impact/etc/zenoss-dsa.properties` configuration file.
For instructions, see [Editing server configuration files with Control Center \(Resource Manager 5.2.x and later\)](#) on page 61.
- 2 To change the number of nodes that Service Impact processes per batch job, increase or decrease the value of `dsa.zenoss.batch_size=100`.
By default, 100 nodes are processed in one update job.
- 3 To change the frequency of targeted updates, increase or decrease the value of `dsa.zenoss.time_period=30`.
By default, the targeted updates occur every 30 seconds as needed.
- 4 After changing values, save the file.
- 5 To activate the changes, restart Service Impact.

Manually rebuilding an impact graph

Service Impact graphs are regularly updated automatically, and rarely require manual actions. However, in unusual circumstances, the Service Impact data store might have inconsistencies that require a manual rebuild of an impact graph to rediscover children of the problem node.

For best performance, do not manually rebuild an impact graph during a time that overlaps a maintenance window. The graph update and maintenance window operations use the same target graph database. Performing them together might lengthen the time to complete the operations, and might result in transaction rollbacks. For more information about manually rebuilding a graph, refer to the *Zenoss Service Impact User Guide*.

Entities that are defined in the Resource Manager object database (ZODB) are normally synchronized and defined in the Service Impact Neo4j datastore. You can manually check consistency between the ZODB and the datastore, and if necessary, rebuild a subset of or all service models.

The manual options allow more granular repair to just a subset of entity impact graphs for all service contexts in which the entity participates. This approach can be much faster than repairing all information. However, if a large amount of entity information must be repaired, rebuilding all with one action might be faster.

To check for consistency between the Resource Manager ZODB and the Service Impact Neo4j datastore, run the following command:

```
zenimpactgraph run --check
```

To update impact dependencies for one or more entity nodes for all service contexts in which they participate, run the following command

```
zenimpactgraph run --update --sb <node_name | node_id | node_path>
--direction <0 | 1> --depth <0 | integer>
```

Commands for `zenimpactgraph run update` are as follows:

- Specify the subgraph by name, path, or ID.

```
--sb=SUBGRAPH, --sub-graph=SUBGRAPH
```

The node path must start with `/zport/dmd/`. Separate multiple nodes with a comma.

- Specify the maximum traversal depth level of the graph to update.

```
--depth=DEPTH
```

If no depth is specified, the default is to update all nodes in the graph, regardless of the number. Specify an integer value to limit the maximum levels of the graph to be updated.

- Specify the traversal direction:

```
--direction=DIRECTION
```

0 indicates `INCOMING`, to discover or rediscover all nodes that impact the target node. If no direction is specified, the default is `INCOMING`.

1 indicates `OUTGOING`, to discover or rediscover all nodes that the target node impacts.

- 1 Log in to the Control Center master host as a user with `serviced` CLI privileges.
- 2 Connect to the `zenimpactstate` container as the `zenoss` user.

```
serviced service attach zenimpactstate su - zenoss
```

- 3 Run `zenimpactgraph run update` with commands.
See the following examples.

Example 1: Basic command using defaults

The following commands use default values to discover nodes for the target node. The target node is specified by name, `vxchnge-vcenter-02.zenoss.loc`. All nodes that impact the target node (incoming direction) and all nodes that the target node impacts (outgoing direction) are discovered. All levels of the graph will be updated (unlimited depth).

```
zenimpactgraph run --update --sb vxchnge-vcenter-02.zenoss.loc
```

Example 2: Discover outgoing nodes for the named subgraph

The following commands discover all nodes that are impacted by the target node, `vxchnge-vcenter-02.zenoss.loc`. All levels of the graph will be updated (unlimited depth).

```
zenimpactgraph run --update --sb vxchnge-  
vcenter-02.zenoss.loc --direction 1
```

Example 3: Discover incoming nodes for the named subgraph

The following commands discover all nodes that impact the target node, `vSphere`, and limit the updates to a depth of 10 levels of the graph.

```
zenimpactgraph run --update --sb vSphere --direction 0 --  
depth 10
```

Example 4: Discover outgoing nodes for the subgraph at the specified path

The following commands discover all nodes that are impacted by the target node, `vxchnge-vcenter-02.zenoss.loc`, and limit the updates to a depth of 100 levels of the graph.

```
zenimpactgraph run --update --sb /zport/dmd/Devices/vSphere/  
devices/vSphere  
--direction 1 --depth 100
```

Example 5: Discover incoming nodes for the subgraph at the specified path

The following commands discover all nodes that impact the target node, `vxchnge-vcenter-02.zenoss.loc`. All levels of the graph will be updated (unlimited depth).

```
zenimpactgraph run --update --sb /zport/dmd/Devices/vSphere/  
devices/vxchnge-vcenter-02.zenoss.loc  
--direction 0
```