

## CLSA-VMDC System Architecture

---

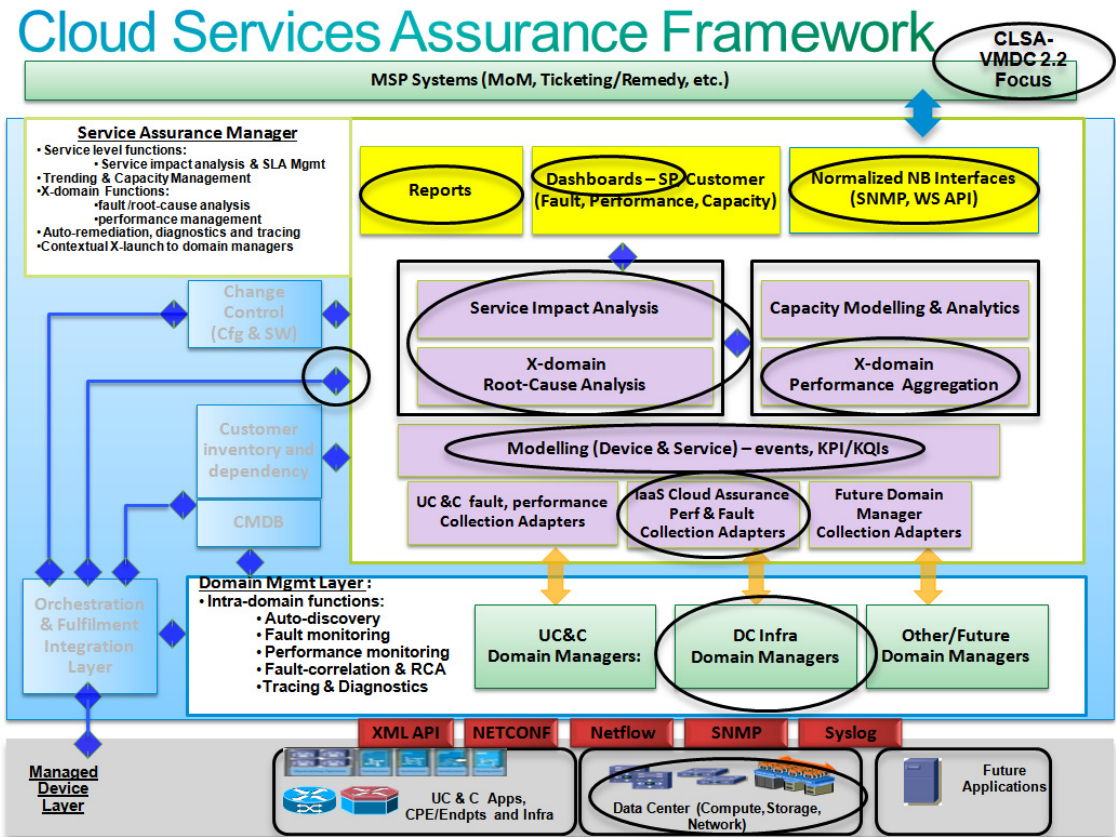
This chapter provides an overview of the Cloud Service Assurance for Virtualized Multi-Services Data Center (CLSA-VMDC) system architecture.

- [“Functional View” section on page 4-1](#) and [“Component View” section on page 4-3](#) provide the functional and component views of the CLSA-VMDC system architecture.
- [“System Components” section on page 4-4](#) defines the components and interfaces used to deliver the system functions.
- [“Monitored Components and Services” section on page 4-5](#) lists the VMDC 2.2 devices that are monitored by CLSA-VMDC.
- [“Key Functions” section on page 4-6](#) defines the functions of the new architecture.

### Functional View

[Figure 4-1](#) illustrates the functional framework for CLSA-VMDC. This functionality is delivered with one or more of the integrated products/components. In CLSA-VMDC, only a subset of this functionality is available. This section defines the functional layers of this architecture and identifies the layers that are available in CLSA-VMDC.

Figure 4-1 Functional View of CLSA-VMDC Architecture



The **Managed Device Layer** consists of Data Center (DC) infrastructure including compute, storage, and network components with instrumentation for inventory, fault, and performance data collection. The instrumentation used in this system includes SNMP, Syslog, XML API, Netconf, vSphere API, and so on. Details of interfaces used per VMDC component are included in “[Monitored Components and Services](#)” section on page 4-5.

The **Domain/Element Management Layer** includes the UCS Manager and vCenter. They provide intra-domain inventory, fault, and performance monitoring for UCS and VMware hosts and VMs. These domain managers offer northbound interfaces APIs as well as SNMP and Syslog interfaces. CLSA-VMDC utilizes UCS XML API and vSphere API interfaces.

The **Service Assurance Manager (SAM) Layer** provides all inter-domain functions and a single pane of glass to monitor all VMDC domains including compute, storage, and network. The high-level functions of each of the SAM layers are as follows:

- **Data Collection Layer.** The collection layer leverages domain managers, third-party tools, and so on to obtain performance, availability, and event data for the end-to-end multi-domain system via a range of open protocols such as SNMP, SSL, WMI, and so on. The collection layer is responsible for normalizing this data into a consistent format and persisting data. Collected data includes inventory, fault, and performance type of information.
- **Modeling Layer.** The modeling layer performs discovery, classification, and modeling to determine component dependencies and service dependency graphs. Both performance and fault data should be included in device and service models.

- **Service Model-based Technology.** CLSA-VMDC uses service model-based technology which is described in more detail in “[Root Cause Analysis and Service Impact Analysis](#)” section on [page 4-15](#) and [Zenoss Cloud Service Assurance Overview, page 5-1](#).
- **Root Cause Analysis (RCA).** Leverages the dependency graph or analytics algorithms to determine which events are the probable root cause of the problem and which ones are just consequences that create noise. Therefore, RCA reduces MTTR. There are a number of different approaches to RCA, but most of them can be classified in one of the following technologies:
  1. Event correlation rules-based
  2. Topology and service model-based
  3. Analytics based
- **Service-Impact Analysis (SIA).** Leverages the dependency graph or analytics algorithms and collects fault and performance data to do the following:
  - Answer who is impacted by the failures
  - Prioritize urgency of failure tickets based on business relevance
  - Determine whether redundancy protected the service
  - Identify failure impacted customers/tenants
  - Prevent future failures by identifying potential service impacting technical risks before they impact service
  - Provide data for SLA measurements and reporting
- **Performance Aggregation Layer.** This layer aggregates performance data from multiple domains (e.g, storage, network, compute for VMDC), normalizes it in the same format and units, provides threshold crossing alerts to the fault management part of the SAM, trends the data over time, and in some cases, performs additional analysis of the data.
- **Presentation Layer.** The presentation layer provides a single view to do both fault and performance monitoring for the entire system. Presentation is done both via dashboards and reports. CLSA-VMDC includes SP dashboards for both fault and performance.
- **Northbound Interface.** The northbound interface is a special form of the presentation layer where normalized and enriched data is presented to northbound OSS/BSS systems via open interfaces such as WS API, SNMP, and email.

## Component View

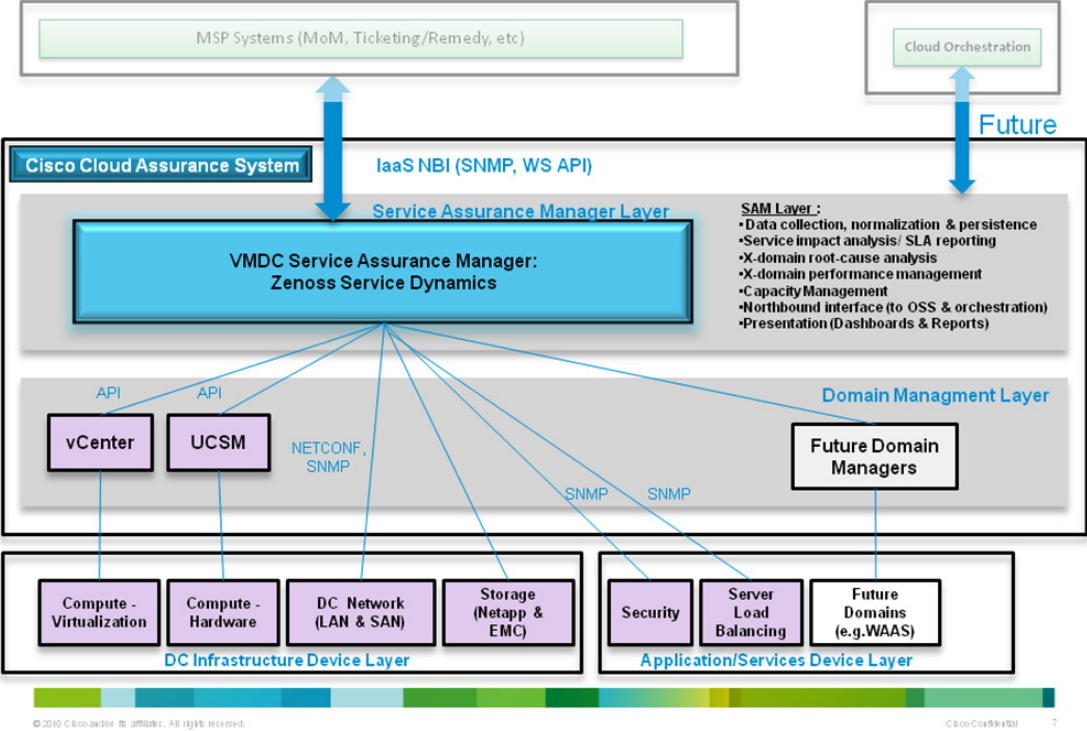
“[Functional View](#)” section on [page 4-1](#) defines the functions of the CLSA-VMDC architecture. This section defines the components used to deliver those functions, as well as their interfaces.

The key component of the architecture for CLSA-VMDC is Zenoss Service Dynamics, which plays the role of the SAM. In addition, two domain managers are utilized - UCS Manager (UCSM) for UCS hardware monitoring, and VMware vCenter for monitoring the virtualized infrastructure.

[Figure 4-2](#) illustrates the components and interfaces used to deliver the functional layers of the CLSA-VMDC architecture.

Figure 4-2 Component View of CLSA-VMDC Architecture

VMDC Cloud Service Assurance Architecture – Component View



Key system interfaces include:

- Southbound Interface instrumentation to collect data from managed system devices.
- Northbound Interface to integrate with OSS/BSS systems such Manager-of-Managers (MoM) (e.g., IBM Netcool), ticketing systems (e.g., Remedy) and so on. The interfaces available from CLSA-VMDC are SNMP, JSON API, email, page, commands, and AMQP interface.
- CLSA-VMDC offers the JSON API interface for integration with orchestration and fulfillment systems.

# System Components

Table 4-1 lists the Cisco and third-party components used in CLSA-VMDC.

Table 4-1 Cisco and Third-Party Components Used in CLSA-VMDC

Vendor	Model	Description
Zenoss	Service Dynamic Resource Manager 4.1.1	Zenoss Service Dynamic software module that performs resource discovery, monitoring, and modeling.
Zenoss	Service Dynamic Impact 4.1.1	Zenoss Service Dynamic software module that performs service impact discovery and analysis.

**Table 4-1 Cisco and Third-Party Components Used in CLSA-VMDC (continued)**

Vendor	Model	Description
Zenoss	Service Dynamic Analytics 4.1.1	Zenoss Service Dynamic software module that performs long term data trending, processing, and reporting.
vCenter	5.0	Domain manager for VMware based virtualization
Cisco	UCSM	Domain manager for UCS platform

## Monitored Components and Services

Table 4-2 lists the VMDC 2.2 devices that are monitored by the CLSA-VMDC system out-of-the-box and the instrumentation (interfaces) used by Zenoss Cloud Service Assurance (Zenoss CSA) to collect data.

**Table 4-2 VMDC 2.2 Components Monitored by CLSA-VMDC**

Managed Component	Component Description	Interface Utilized by Zenoss
<b>Infrastructure - Compute</b>		
UCS 5108; B200 M1 Blades, M2 Blades	Unified Compute System and Server Blades, v1.4(2b)	UCSM XML API
UCS C210/C200s		UCSM XML API, SNMP
UCS 6120 and 6140		UCSM XML API
VMware ESXi Hypervisors	5.0	vSphere API
VMware Virtual Machines		vSphere API
<b>Infrastructure - Storage</b>		
MDS 9513	SAN Switch, NX-OS v5.0(4d)	SNMP
EMC VMAX	Storage Array	AS engagement required, out-of-box support in next release of CLSA-VMDC
NetApp FAS6080, FAS3k	Storage Array	SNMP, SSH
<b>Infrastructure - Network</b>		
Nexus 1000V / Nexus 1010	Distributed Virtual Access Switch, NX-OS v4.2(1) SV1(4)	NETCONF, SNMP
UCS 6100	DC Access Switch	SNMP, UCSM XML API
Nexus 5000 - 5548, 5596, and 5520	DC Access Switch 5.1(3)N1(1a)	NETCONF, SNMP
Nexus 7018	DC Aggregation/Access, NX-OS v5.2 aka "Delhi"	NETCONF, SNMP
Nexus 7010	DC Core, NX-OS v5.2	NETCONF, SNMP
Nexus 2200	1 G Fabric Extender, NX-OS v5.2	NETCONF, SNMP

**Table 4-2 VMDC 2.2 Components Monitored by CLSA-VMDC (continued)**

Managed Component	Component Description	Interface Utilized by Zenoss
ASR 9000	DC WAN Edge/PE Router, Software release 4.1.2	SNMP, SSH
ASR 1000	IP NGN PE, IOS 15.2(1)S1	SNMP
<b>Network Services Components</b>		
Catalyst 6500 VSS	DC Services Node	SNMP, SSH
ACE20 and ACE30	Application Control Engine Module A2(3.3)	SNMP, ACE XML API
FWSM	FWSM Firewall Service Module for 65004.1(4)	SNMP
ASA 5580-40	RA IPSec and WebVPN Gateway, v8.4.3	SNMP
ASA 5585-40	Multi-Context Firewall Appliance (v8.4.3)	SNMP
Virtual Security Gateway	Virtual Firewall (vApp appliance), v4.2(1) VSG1(3.1a)	SNMP, NETCONF, SSH

## Key Functions

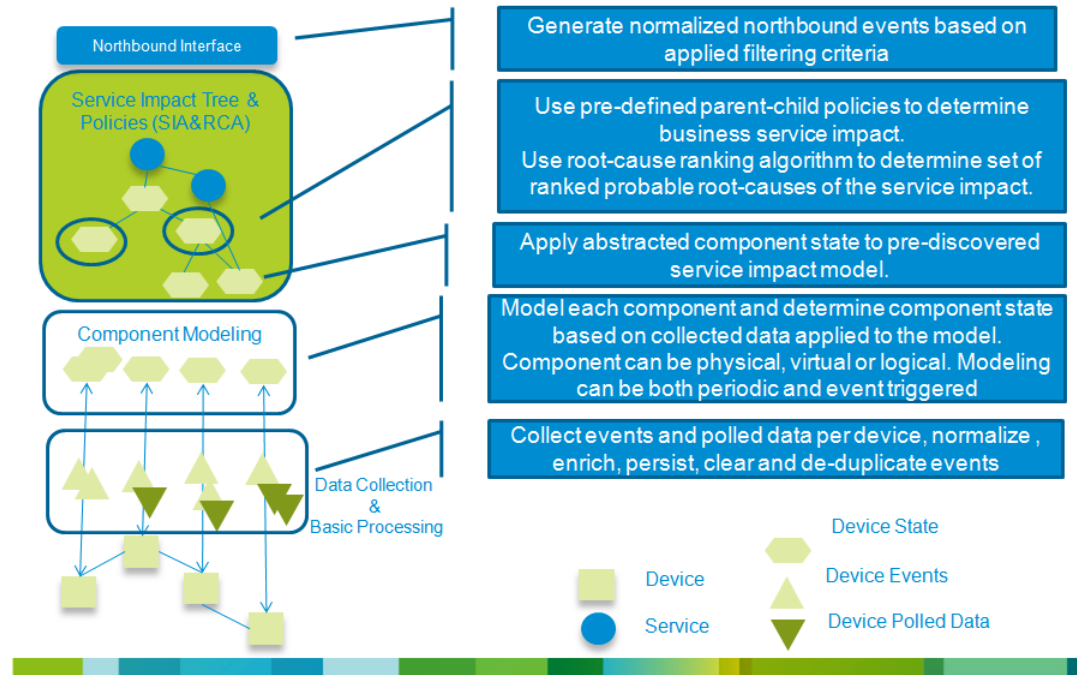
This section describes the key functions of CLSA-VMDC.

In the overall lifecycle of assurance services, the first task that has to be completed is enablement of service assurance services. [“Auto-Enablement of Service Assurance” section on page 4-8](#) provides details about enabling service assurance, including provisioning and auto-discovery. Once assurance services are enabled, they can be used for Day 2 operations. [Figure 4-3](#) illustrates and explains the high-level, end-to-end data flow through the fault and problem management part of CLSA-VMDC.

Figure 4-3 End-to-End Fault and Problem Management Data and Processing Flow

## High Level Data & Processing Flow

### Service topology modeling based integrated RCA & SIA



The following sections discuss each of the stages and functions in this sample data flow, including:

- [Auto-Enablement of Service Assurance](#), page 4-8
- [Fault, Performance, and Configuration Data Collection and Device Modeling](#), page 4-11
- [Event Processing](#), page 4-14
- [Root Cause Analysis and Service Impact Analysis](#), page 4-15
- [Northbound Interface](#), page 4-21

This section also discusses a few additional functions related to the overall platform and its use, including:

- [Performance Management](#), page 4-28
- [Dashboards](#), page 4-29
- [Reporting](#), page 4-34
- [Multi-Tenancy](#), page 4-36

## Auto-Enablement of Service Assurance

Auto-enablement of service assurance can be achieved in a couple of different ways. Fundamentally, the following are approaches that can be taken to automate service enablement and life cycle:

1. Reduce necessary amount of configuration (by using technology that is self learning (e.g., self learning thresholds))
2. Auto-discovery (by assurance system)
3. Programmatic orchestrated provisioning (via integration with orchestration system)

CLSA-VMDC focuses on auto-discovery. CLSA-VMDC also provide APIs for programmatic orchestrated provisioning, but they are not integrated or validated with any particular orchestration system. Auto-discovery and APIs are discussed in the following sections.

### Auto-Discovery

The following types of objects are auto-discovered in CLSA-VMDC:

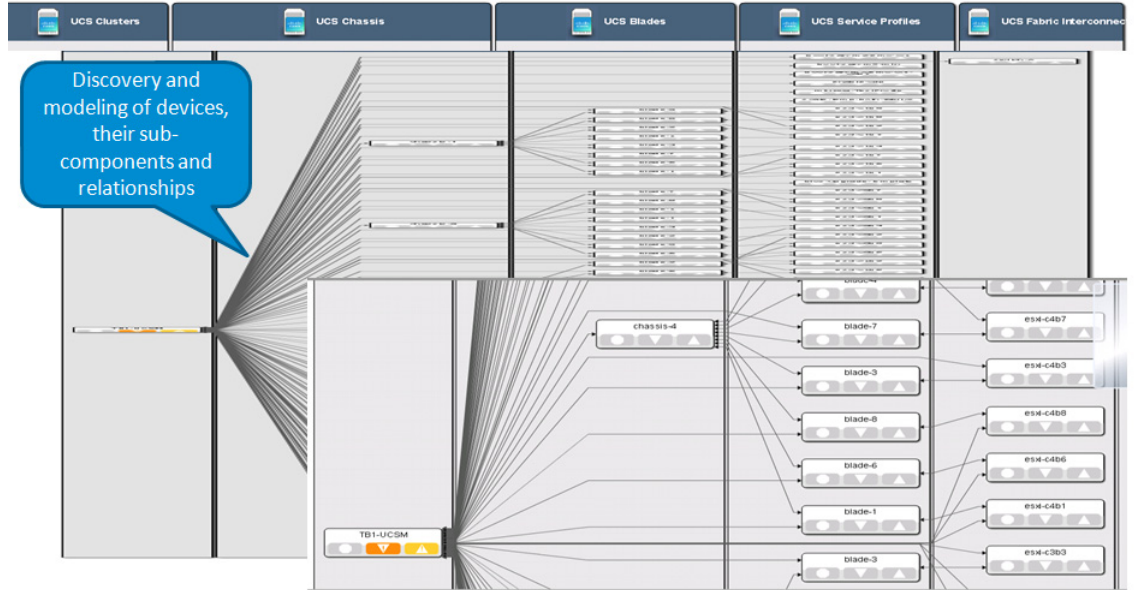
- Monitored devices (e.g., UCS, Nexus 7000, MDS 9000, etc.)
- Sub-components of devices and their relationships (e.g., UCS chassis, blades, fabric interconnect, etc.)
- Tenant-based Service Impact Analysis (SIA) model for the compute (e.g., tenant Virtual Machine (VM) mapping to service impacting resources, both dedicated and shared vCenter and UCSM managed resources). The exception is tenant name and its link to the service, which cannot be discovered, but relies on orchestrated provisioning. In this release, tenant name and mapping to the VM are provisioned manually, but the API is provided.

[Figure 4-4](#) and [Figure 4-5](#) illustrate examples of auto-enablement of service assurance.

Figure 4-4 Real-time Auto-discovery of Device Components—Cisco UCS

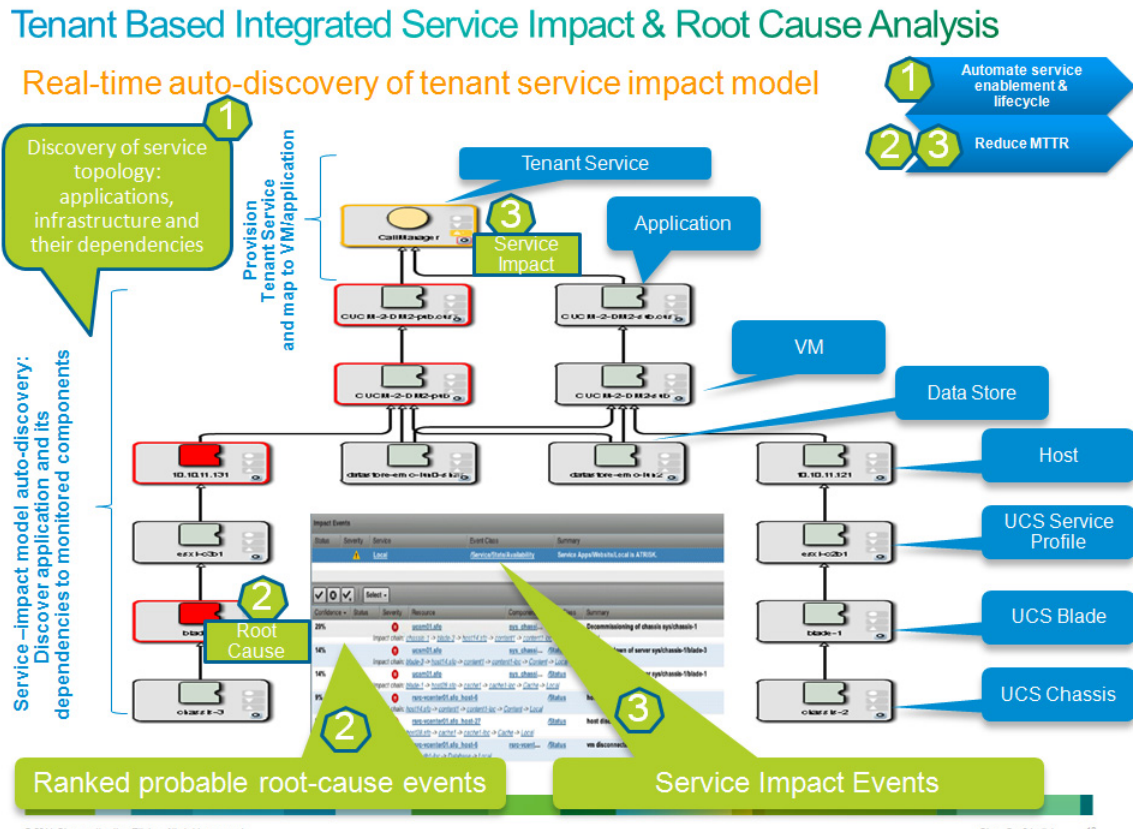
# Auto-enablement of Service Assurance Example

## Real-time auto-discovery of device components – Cisco UCS



Source: Zenoss Service Dynamics

Figure 4-5 Real-time Auto-discovery of Tenant Service Impact Model



## Zenoss APIs for Programmatic Provisioning

CLSA-VMDC offers APIs to programmatically provision the following components in the service impact tree:

- Tenant Name
- Tenant ID
- Service Name
- Service ID
- VM Name
- VM ID

This enables automatic on-boarding of the tenant and tenant compute service, which maps them to the already auto-discovered VM and its relationships to shared hardware.

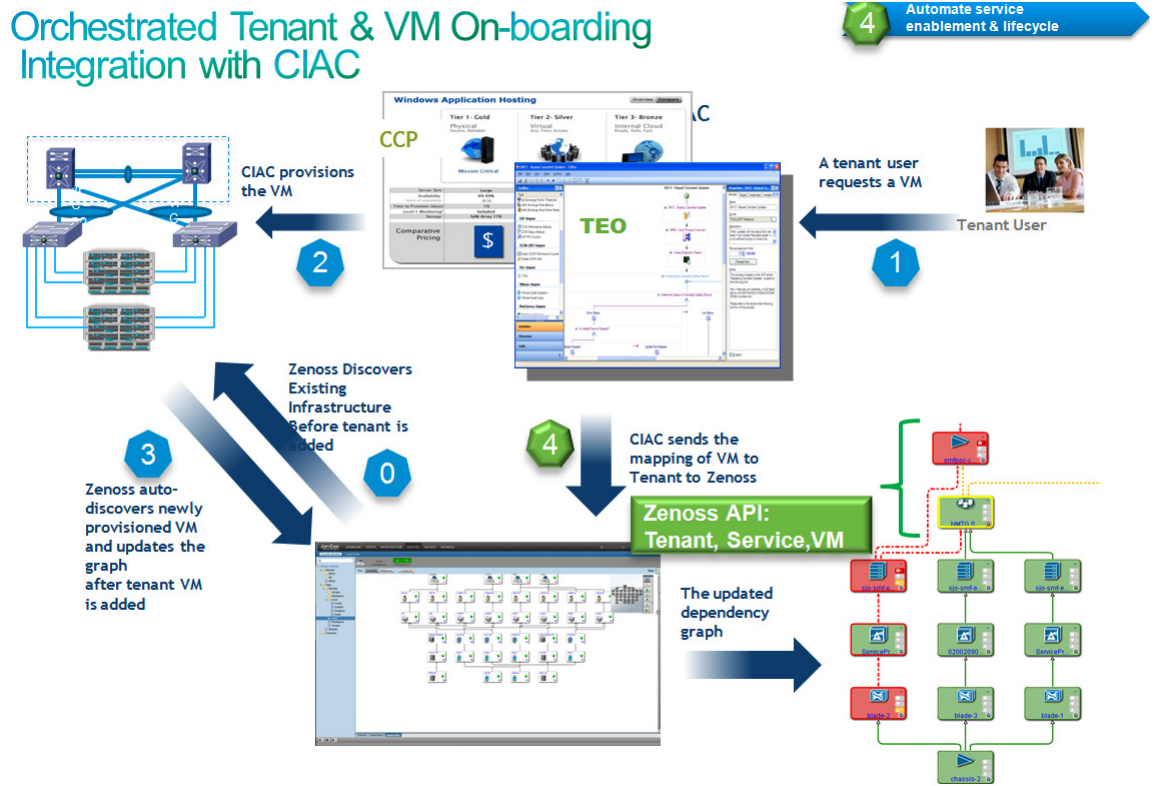


### Note

Proof of Concept of this functionality integrated with the Cisco Intelligent Automation for Cloud (CIAC) orchestration stack has been performed by Cisco Advanced Services; however, it was not validated as part of the CLSA-VMDC system. If this functionality is desired in the field before it is included as part of the Systems Development Unit (SDU) system release, then Cisco Advanced Services can perform integration with the desired orchestration stack using the provided API.

Figure 4-6 illustrates the high-level workflow that provisions the tenant and tenant service and then maps the workflow to the auto-discovered VM and the rest of the auto-discovered infrastructure.

Figure 4-6 Zenoss Tenant Provisioning Using CIAC Orchestration



## Fault, Performance, and Configuration Data Collection and Device Modeling

Consolidated monitoring and data collection at the SAM layer is one of the fundamental functions of CLSA-VMDC. Consolidated monitoring and data collection is characterized by the following attributes:

- Various domains (applications, compute, storage, network). The cloud assurance system needs to provide single pane of glass to monitor components from various domains.
- Fault and performance data. The cloud assurance system needs to consolidate fault and performance data and leverage both for all of its higher order functions like RCA and SIA.
- Various data sources, interfaces, and protocols. The cloud assurance system needs to collect data from multiple data sources and protocols and consolidate this data in unified device and service models. Some examples of different data sources and protocols are SNMP, syslog, WS API, Netflow, customer opened tickets, and so on.

### Zenoss Data Collection

Zenoss CSA offers consolidated monitoring for VMDC, including consolidation of domains (i.e., support for OS, compute, storage, and network), consolidation of performance and fault data (i.e., takes into consideration both polled performance data, asynchronous events it receives, as well as synthetic

events it generates for both performance and availability), and consolidation of data sources (i.e., device monitoring models utilize multiple data sources such as SNMP, syslog, API, and consolidate it within unified device model).

Zenoss Service Dynamics uses an agentless data collection approach, which is critical for the type of scale expected in cloud systems. Instead of installing an agent on monitored devices, Zenoss supports a rich set of protocols to enable data collection. A list of protocols used for data collection from VMDC devices is included in “[Monitored Components and Services](#)” section on page 4-5. The following is a more comprehensive list of data collection interfaces that the Zenoss CSA platform supports:

Event input:

- SNMP
- Syslog
- XML Remote Procedure Call (XMLRPC)
- JavaScript Object Notation (JSON)/API
- Advanced Message Queuing Protocol (AMQP)

Easily configurable protocol usage:

- Secure Shell (SSH)
- Java Management Extensions (JMX)
- Windows Management Instrumentation (WMI)
- Perfmon
- Any script that returns data in a known format (such as Nagios)

Other collection mechanisms (model/performance/event data):

- Internet Control Message Protocol (ICMP)
- Telnet
- JMX
- Hypertext Transfer Protocol (HTTP) - Web Transactions
- Oracle
- Structured Query Language (SQL) Server
- MySQL
- Apache (mod\_status)
- memcache
- Splunk Queries
- Simple Mail Transfer Protocol (SMTP)
- Post Office Protocol (POP)
- UCSM XML API
- vSphere Simple Object Access Protocol (SOAP) API
- vCloud Director
- Amazon EC2 and CloudWatch
- Cisco CallManager (AXL)
- Domain Name System (DNS)

- Lightweight Directory Access Protocol (LDAP)
- Network Time Protocol (NTP)
- File Transfer Protocol (FTP)
- Internet Relay Chat (IRC)
- Extensible Messaging and Presence Protocol (XMPP)
- Remote Procedure Call (RPC)
- Network News Transfer Protocol (NNTP)

## Zenoss Device Modeling

Device modeling in Zenoss goes beyond traditional device discovery; it also uses standard Management Information Bases (MIBs) to discover interesting aspects of the device and automatically defines models for that device type. Once modeled, these learned attributes can be inherited as part of the model when a new device of the same type is discovered again. The information below describes various attributes of the Zenoss device modeling process.

Initial Zenoss Model (plugins):

- Interfaces to access device and objects of interest (KPIs, events, thresholds, etc.) are statically defined
- Models are assigned to a device class

Device Modeling:

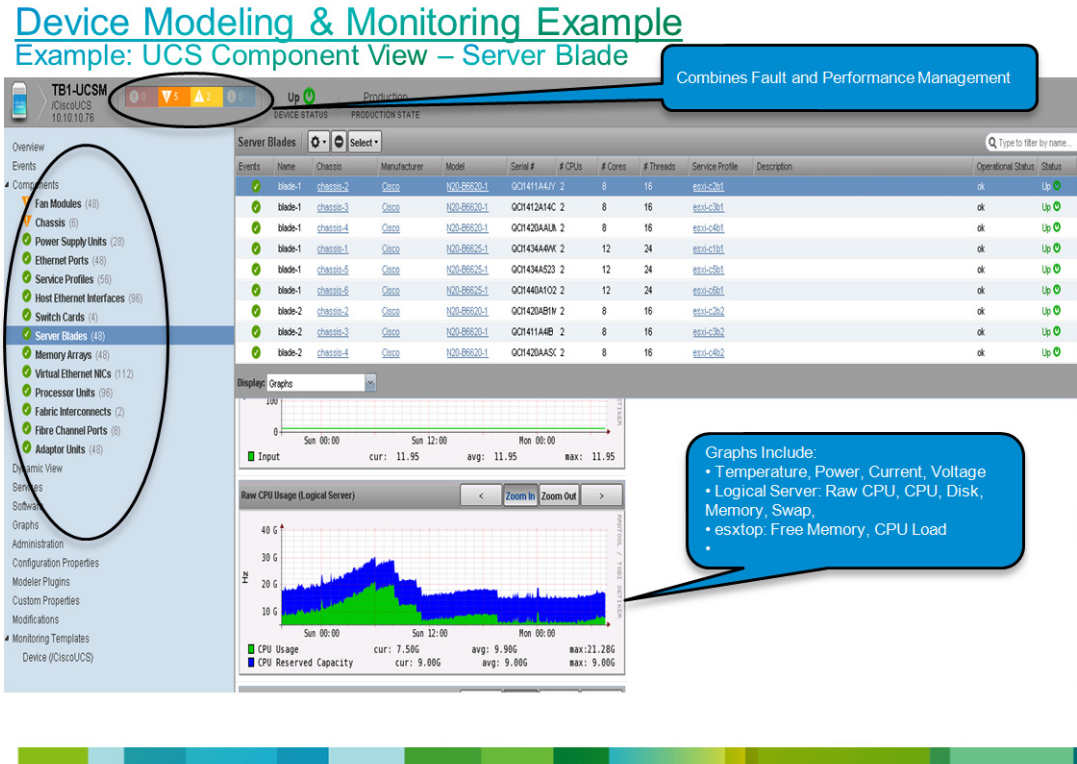
- During individual device discovery, all modeler plug-ins for the device class are automatically considered, and a model per instance of the device is created.
- After discovery modeling, monitoring and event processing automatically starts.

Device Remodeling:

- Model per device instance can dynamically change in response to events (e.g., blade removed, etc.)
- ZenModelerDeamon - per collector configuration happens every 12 hours
- ZenVMwareDeamon (exception for VMware and remodels every 4 hours)
- List of events that trigger remodeling is configurable (default set exists)

An example of unified monitoring using Zenoss CSA is illustrated in [Figure 4-7](#).

Figure 4-7 Unified Monitoring Using Zenoss CSA



## Event Processing

In CLSA-VMDC, event processing is divided into two categories:

- Basic event processing
- Event processing that is part of RCA and SIA

This section only describes basic event processing functions, while RCA and SIA are discussed in the following sections. The basic event processing functions included in this system are event classification, normalization, de-duplication, enrichment, persistence, and clearing.

- **Event classification** groups similar events in event classes, so that some of more complex processing may be simplified by looking at event classes rather than each individual event.
- **Event normalization** translates various formats of the raw collected data into a single format that is used by the SAM. Often, the same format or subset of the fields of normalized format can be sent to northbound systems. This function allows simplified integration of northbound systems since they have to deal with a single event format for multiple device types and instrumentation protocols.
- **Event de-duplication** eliminates multiple events that have the exact same content with exception of time stamp. After de-duplication, a single event is kept, and typically a counter indicating number of occurrences of the event is added, as well as a timestamp indicating the first and last occurrence of the duplicate event.
- **Event persistence** archives all events to be used for forensic analysis. In some systems, persistence exists only on post-processed events, while in others, for raw events as well.

- **Event clearing** is used to indicate when the original condition for which the event was raised is removed. Explicit event clearing is done by generating clearing events with the field within the clearing event, which points to the ID of the event that it is clearing. For example, if an interface down event for a specific interface had an ID of ID1, when the interface goes up again, an event with ID2 should be raised, which includes as one of its fields a reference to event ID1. Explicit event clearing is recommended. In addition to explicit clearing, time-based clearing can be utilized as well. Time-based clearing clears the event after a specific time interval elapses from the time that the original event was received.

## Root Cause Analysis and Service Impact Analysis

One of the key functions of CLSA-VMDC is Root Cause Analysis (RCA) and tenant-based Service Impact Analysis (SIA).

The objective of RCA is to reduce MTTR by determining which events are probable root causes of the problem and which events are just consequences that create noise.

The following are the objectives of tenant-based SIA:

- To prioritize the urgency of failure tickets based on business relevance
- To determine whether redundancy protected the service
- To identify failure impacted customers/tenants
- To prevent future failures by identifying potential service impacting technical risks before they impact service
- To enable Service Level Agreement (SLA) measurements and reporting

## Zenoss SIA and RCA

Zenoss Service Dynamics uses model-based SIA, which produces a set of ranked probable root causes as a by-product of SIA. This service impact-based approach to RCA is a fundamentally different approach from legacy rule-based systems:

- Bottom-up. What services are impacted by conditions below (Zenoss) vs.
- Top-down. What is the cause of problem at service level (legacy products)

Zenoss does not determine a single root cause, but instead identifies multiple related events (probable root cause events) and presents the following:

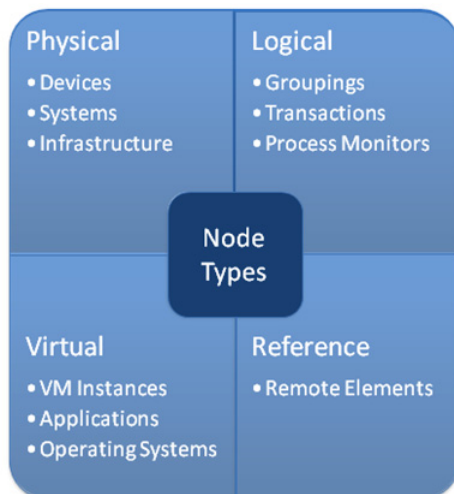
- A root cause ranking algorithm is utilized to rank probable root cause events in order of confidence that the event is the actual root cause event. This algorithm ranks impact events based on a variety of criteria, including the severity of the event, service graph depth, and the number of graph branches affected by an event.
- Hierarchical service dependency graphs provide a visual indication of probable root causes leading to a service impact.

Events flow through the graph referencing molecular node policies to determine whether they should be passed, filtered, aggregated, or masked. There are a few key elements of RCA and SIA in Zenoss Service Dynamics. Each assurance service within Zenoss is modeled with a service impact tree that consists of a set of nodes, policies applied to the nodes, and the relationships between the nodes:

- The service can be arbitrarily defined and can be a very abstract service that consists of other sub-services, or on other extreme, one can even define a single physical interface as a service. This provides a very flexible framework for service definition.

- Model nodes represent arbitrary components such as physical, logical, or virtual resource. For example, nodes can represent an end-to-end service such as voice, a virtual resource such as a VM, or a physical resource such as a chassis or physical interface. The following four types of nodes are currently supported, as illustrated in [Figure 4-8](#):
  - Physical. Systems, infrastructure, and network devices that a service relies on.
  - Virtual. Software components that make up a service.
  - Logical. Aspects of a service that must be measured or evaluated as a set to determine state (facilitates extension of an impact graph by providing a hook to incorporate arbitrary events into impact analysis).
  - Reference (future release). Provide a link to dependencies managed by an external instance of Zenoss or other management system capable of propagating state information to Zenoss.

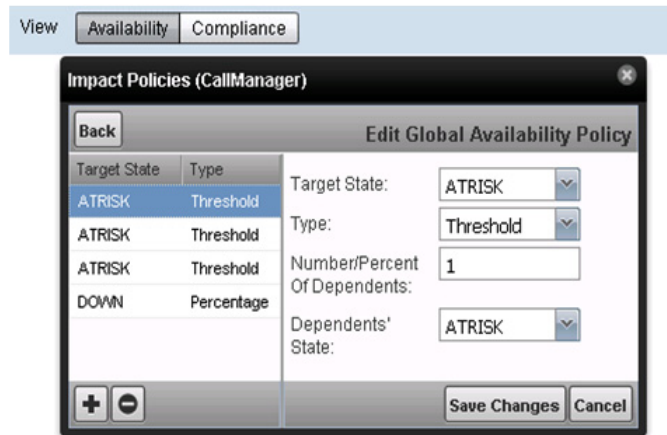
**Figure 4-8 Node Types**



- Policy is defined per node, which allows it to move as the resources move, which is a critical characteristic for the cloud environment. Zenoss refers to this policy as a molecular policy since it is defined per node. Zenoss utilizes a very simple policy that can define the state of the node solely as a function of the state of its children nodes, which allows for service impact "rules" decoupling from device events resulting in the following:
  - "Rules" defined in a single place for any given device or service: device events processing in event processing software modules, service impact processing in service impact graphs (i.e., device events do not need to be considered in service level rules)
  - Simplified development and maintenance of cross-domain service impact and RCA customizations: do not have to correlate device events from multiple devices to determine cross-domain service impact and possible root causes
  - Note that whenever desired, device events can be used as part of service impact "rules" via use of logical nodes whose rules define how to interpret the service impact of specific events based on its type and severity.
  - Policy can be global or contextual:
    - Global policy applies to device/service type in any service graph.
    - Contextual policy applies only to device/service in the particular service graph.

- Each node has a default policy applied, which reduces the need for custom configuration. The default policy is often sufficient, but can be modified where required via GUI or API. [Figure 4-9](#) illustrates a sample node policy.

**Figure 4-9 Sample Node Policy**



- Each node has a state which is determined using node policy. Currently, there are four service and node availability states supported (UP, DOWN, DEGRADED, AT RISK), and two service performance states supported (IN/OUT). Software development is required to increase the number of states.

Creation of the service model in Zenoss Service Dynamics can be done in the following ways:

- Auto-discovery for set of service models predefined by Zenoss. Examples are service models for OS, VMware, and UCS domains.
- A GUI service model can be modified or created via a simple GUI approach.
- REST APIs allow for programmatic provisioning of the service models.

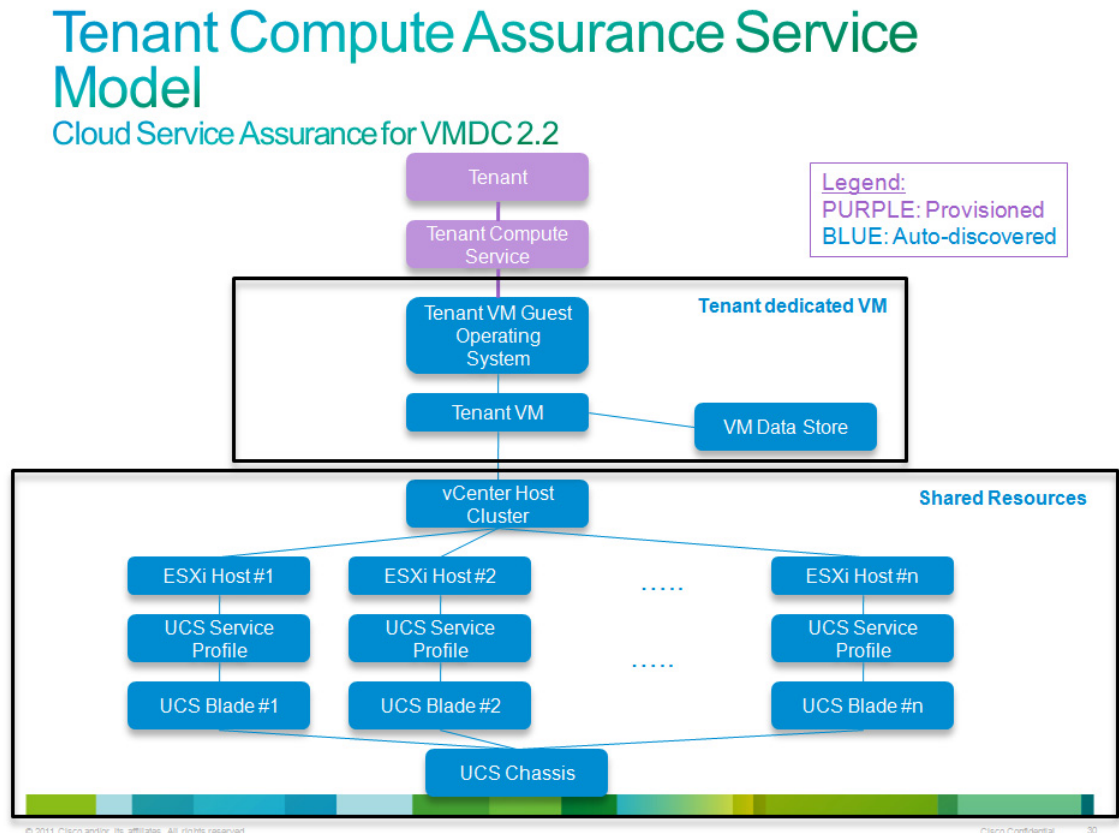
## VMDC Assurance Service Models

To perform SIA, CLSA-VMDC uses service models with polled and asynchronous data to perform SIA and RCA. CLSA-VMDC offers an out-of-the-box tenant service model for compute. In future releases, CLSA-VMDC will expand the library of out-of-the-box service models that will be validated and delivered as part of this system. However, note that users can easily customize service models as well as create new ones.

### Tenant Compute Assurance Service

[Figure 4-10](#) defines the out-of-the-box tenant compute service model to be delivered as part of CLSA-VMDC.

Figure 4-10 Tenant Compute Assurance Service Model—Generic Application



### Service Model Policy

Each node (referred to as the parent node) in the service model has a policy defined that calculates the state of that node based on the state of its children and any explicit events associated with the parent node.

For the particular service model illustrated in Figure 4-10, the specific policies listed in Table 4-3 should be applied.

**Table 4-3 Service Model Policy Decisions**

Node	Node State	If Child Node State
Tenant Compute Service	UP/DOWN/AT RISK	UP/DOWN/AT RISK
Tenant Guest OS	UP/DOWN/AT RISK	UP/DOWN/AT RISK
Tenant VM	UP/DOWN/AT RISK	UP/DOWN/AT RISK
ESXi Cluster	<ul style="list-style-type: none"> <li>• UP/DOWN</li> <li>• AT RISK</li> </ul>	<ul style="list-style-type: none"> <li>• All Children UP/DOWN</li> <li>• At Least One Child DOWN/AT RISK</li> </ul>
ESXi Host	UP/DOWN/AT RISK	UP/DOWN/AT RISK
UCS Blade	UP/DOWN/AT RISK	UP/DOWN/AT RISK

Out-of-the-box, all nodes use the default policy where worst impact wins. The one exception is the VMware cluster, which is DOWN if all children are DOWN, and DEGRADED if any nodes are DOWN or DEGRADED.

In addition to considering the parent child policy, explicit state of the nodes is determined by availability and events for components node represents. For VMware and UCS nodes, the explicit node impact status is determined mainly by modeled properties. As modeling occurs or various events are received, Zenoss reassesses the impact state by querying the Zenoss model. For example, when a VM power off event is received, the model is updated and the VM status is reassessed and updated.

### Service Model Variations

Note that the model defined in this section illustrates a single-tier application with a single VM. Variation of this service model would be models for the following:

- Multi-tier application, where there would be multiple "tenant dedicated VM" blocks tied to the tenant compute service. The tenant compute service default policy may need to be customized.
- Single-tier application that supports application level redundancy via clustering (e.g., Cisco UC applications such as CUCM). In this case, the model would be modified to include multiples of "tenant dedicated VM" blocks. The default policy used for the "tenant compute service" should be applicable. An example of this service model is illustrated in Figure 52365604-001.

### Service Model Enablement

Most of this model is auto-discovered, while the top node of the service model needs to be provisioned. Typically, provisioning would be done in an automated way when the tenant and VM get on-boarded. In CLSA-VMDC, there is no integration with the orchestration stack, and as such, the top node of the service model is manually provisioned. Note that in real deployments, per-tenant manual provisioning is not an option, in which case either an available Zenoss API can be used by the orchestration platform of choice, or if not provisioned, the tenant service impact is still possible but results will be given in the VM context rather than tenant service context. For example, there would be no automatic mapping between tenant name, tenant service name, and VM ID.

In future CLSA-VMDC releases, integration with VMDC orchestration stacks will be implemented and validated. In addition to auto-discovery of the service model from VM down, if operating systems such as Windows or Linux are deployed, they should also be auto-discovered.

### Mobility Handling

The host to VM relationship is given by VMware during modeling stage. Whenever VMware generates an event that indicates VM movement, Zenoss reacts and remodels the source and target hosts to update its model. Depending on the event collection interval specified in the Zenoss configuration, the model change can take anywhere from 15 seconds to 3 minutes. With the out-of-the-box configuration, the average time would be about 1 minute.

### Redundancy Implications

A service model with three service states accounts for redundancy. The AT RISK state is used to indicate conditions where the service or service model node is still functioning despite a failure of one of its children because redundancy protected the service. For the particular service model shown in Figure 52365604-001, redundancy capabilities that are accounted for include the following:

- If one of the blades/hosts fails, and the vCenter cluster that VM belongs to has multiple blades/hosts, then the VM node will be marked AT RISK as opposed to down based on the status of its children. Note that explicit VM related state and events can result in the state of the VM node being down even though the state of its children alone would result in an AT RISK state
- In a case where there is application level redundancy and thus more than one VM and application deployed for single tier applications, there is also service model redundancy built in on the application/VM level. For example, a service is AT RISK if one of the application nodes/VMs is DOWN because the remaining application/VM nodes provides redundancy for the failed application/VM node.

## VMDC RCA and SIA Use Cases

Once service impact models are defined, the data is applied to service impact models to maintain real-time state of the service availability and performance, as well as to determine probable root cause of any failures that may happen.

This section provides a list of failure scenarios (use cases) validated as part of the CLSA-VMDC test effort, for which the out-of-the-box compute service model can determine correct probable root cause and service state for previously defined services. All of the use cases are validated in an environment where VMware High Availability (HA) is deployed.

Use Case Name (Fault):

- VM Failure
- VM vNIC failure<sup>1</sup>
- VM vMotion<sup>2</sup>
- ESXi host failure
- UCS Blade failure
- UCS chassis failure
- UCS P/S failure
- UCS FEX failure
- UCS 6100 chassis failure
- UCS 6100 interfaces to UCS 5100 failure
- VM CPU degradation (Threshold Crossing Alert (TCA))
- VM Memory degradation (TCA)
- Host CPU degradation (TCA)
- Host Memory degradation (TCA)

1. VM vNIC failure requires event transform customization.

2. VM vMotion is not a true fault event, since the VM stays up; however the impact graph does track the VM's host swap.

## Northbound Interface

One of the key, new functions of CLSA-VMDC architecture is a single, normalized Northbound Interface (NBI) provided by the SAM.

The key objectives of the single, normalized interface are:

- To simplify and reduce cost of integrating providers existing northbound system in the CLSA-VMDC system. The provider needs to integrate and maintain just one interface rather than multiple dozens of interfaces towards individual devices and/or domain managers. CLSA-VMDC will be responsible for absorbing updates related to any relevant changes in the underlying system and devices.
- To enable CLSA-VMDC to be inserted in various business and operational deployment environments. This is achieved by offering a variety of interface protocols, rich filtering capabilities, and notifications tenant awareness.
- To enable CLSA-VMDC to simplify service assurance of overlaid application based systems that are deployed on top of VMDC infrastructure. An example of this type of system is the Hosted Collaboration Solution (HCS). This is achieved by providing tenant service level notifications rather than device level notifications, which enables service overlay (or multi-tier SIA) to be implemented by HCS, and as such, Cloud Service Assurance-HCS (CLSA-HCS) would have to deal with the state of only a handful of services coming from CLSA-VMDC, rather than thousands of events coming from individual VMDC devices.

Zenoss northbound integration is supported via:

- JavaScript Object Notation (JSON)/Representational State Transfer Application Programming Interface (ReST API)
- SNMP traps (ZENOSS-MIB.txt)
- Syslog
- Event queues (AMQP, Java/Python wrappers) and event commands (command line call with event context)

Configurable filtering capabilities are offered to provide different data to different northbound consumers. The following sections describe the interfaces, data, and filtering capabilities in more detail.

- [SNMP Northbound Interface, page 4-21](#)
- [Zenoss SNMP Notification Content, page 4-22](#)
- [Zenoss Notification Filtering, page 4-23](#)
- [WS or ReST API, page 4-24](#)
- [Northbound Integration Use Case Examples, page 4-24](#)

## SNMP Northbound Interface

One of the key requirements for CLSA-VMDC is to offer asynchronous notifications via Simple Network Management Protocol (SNMP). These notifications are consumed either by the provider's existing northbound systems such as MoM, ticketing, and SLA management systems, or by other Cisco systems deployed on VMDC architecture such as HCS.

Regardless of the source or type of the event, all events should be sent using the same normalized format. However, as discussed in this chapter, there may be differences in the values of the populated fields based on the type of events (e.g., service impact events contain information about service name and state, while device level events do not).

## Zenoss SNMP Notification Content

Zenoss CSA uses a custom Zenoss Management Information Base (MIB) for northbound notifications. The existing MIB addresses the resource manager part of the product, but not the service impact part. MIB extensions will be designed to address service impact events and related probable root cause events as a part of the next phase of CLSA-VMDC. In the interim, implementation that has been validated and included with the CLSA-VMDC release includes scripts provided by Zenoss that generate service impact events and related probable root cause events in a format consistent with the format used internally within Zenoss products. Events associated with devices use existing Zenoss MIB for notifications.

The ZENOSS-MIB.txt is located in the following Zenoss directory: \$ZENHOME/share/mibs/site. See [Enable Event Forwarding](#) for more information.

In summary, for the CLSA-VMDC release:

- Device level events use the Zenoss MIB format specified in Table 52144939-001.
- Service events and related events use the internal event format generated using the provided script.
- SNMP notifications can be sent to multiple destinations.

The following document contains useful information related to configuring notifications:

<http://community.zenoss.org/docs/DOC-12101>



### Note

Both ZENOSS-MIB and service impact notification scripts were validated as part of CLSA-VMDC.

[Table 4-4](#) maps the fields of existing Zenoss MIBs to the SAM requirements.

**Table 4-4 Zenoss MIB Fields**

Zenoss MIB Field Name	Description
evtId	Unique identifier ID of the event
evtDedupid	De-duplication ID of the event
evtDevice	Device associated with event
evtComponent	Device component associated with event
evtClass	Event classification
evtKey	Event key used for refining event granularity beyond device and component. Used in de-duplication, auto-clearing.
evtSummary	Event message truncated to 128 characters
evtSeverity	Event severity number: 0=clear(normal), 1=debug, 2=info, 3=warning,4=error, 5=critical
evtState	Event state number: 0=new, 1=acknowledged, 2=suppressed
evtClassKey	Class key for rule processing often matches component
evtGroup	Logical grouping of event sources
evtStateChange	Last time event changed through administrative activity
evtFirstTime	First time an event was received
evtLastTime	Last time an event was received
evtCount	Number of times this event has been seen

**Table 4-4 Zenoss MIB Fields (continued)**

Zenoss MIB Field Name	Description
evtProdState	Production state of the device or component associated with this event
evtAgent	Collector process that received or created this event
evtDeviceClass	Class of device that this event is associated with
evtLocation	Location of device that this event is associated with
evtSystems	Systems containing the device that this event is associated with
evtDeviceGroup	Groups containing the device that this event is associated with
evtIpAddress	IP address that this event was generated or sent from
evtFacility	Syslog facility if the event was initially sent as a syslog
evtPriority	Syslog priority if the event was initially sent as a syslog
evtNtEvId	Windows NT_EVENT_ID if the event was initially received from Windows event log
evtOwnerId	User that acknowledged this event
evtClearId	evtId that cleared this event
evtDevicePriority	Priority of the device that this event is associated with
evtClassMapping	Name of the event class mapping that matched this event

The following fields are available in the script generated notification for the service impact events:

- Service name
- Severity
- Timestamp
- Service state
- URLs to EventDetail, page to acknowledge and close events, device events
- All events in the impact chain. Each event in the impact chain includes:
  - Device
  - Component
  - Device Class
  - Event Class
  - Severity
  - Timestamp
  - Message
  - URLs to EventDetail, page to acknowledge and close events, device events

## Zenoss Notification Filtering

Filtering capabilities can be used to customize notifications based on the needs of different northbound consumers:

- There is a capability to have multiple notification subscribers/receivers.

- Each notification subscriber/receiver should be able to apply different filter, e.g., one receiver may subscribe to service events while the other may subscribe to root cause events, or one receiver may subscribe to compute events while the other may subscribe to network events.
- Each system user should be able to apply different filters.

For more information, see section 6.2.3.1 in [Working with Triggers and Notifications](#).

## WS or ReST API

The JSON API can be used to obtain the following:

- Device model and attributes data
- Performance data
- Event data
- Service data

Most of the information visible via the GUI can also be obtained via the JSON API.

In addition to retrieving data, the JSON API can also be used for the following:

- Managing events (acknowledge, clear, close)
- Adding devices to be monitored
- Setting production state
- Initiating discovery and modeling of devices
- Managing thresholds
- Managing reports
- Other configurations

More information on the JSON API can be found at the following URL:

[http://community.zenoss.org/community/documentation/official\\_documentation/api](http://community.zenoss.org/community/documentation/official_documentation/api)

## Northbound Integration Use Case Examples

This section includes typical use cases that illustrate the rich filtering capabilities of the NBI.

- [Abstraction via Single Interface, page 4-24](#)
- [Integration With Multiple Northbound Systems, page 4-25](#)
- [Abstraction Through Service Overlays, page 4-27](#)

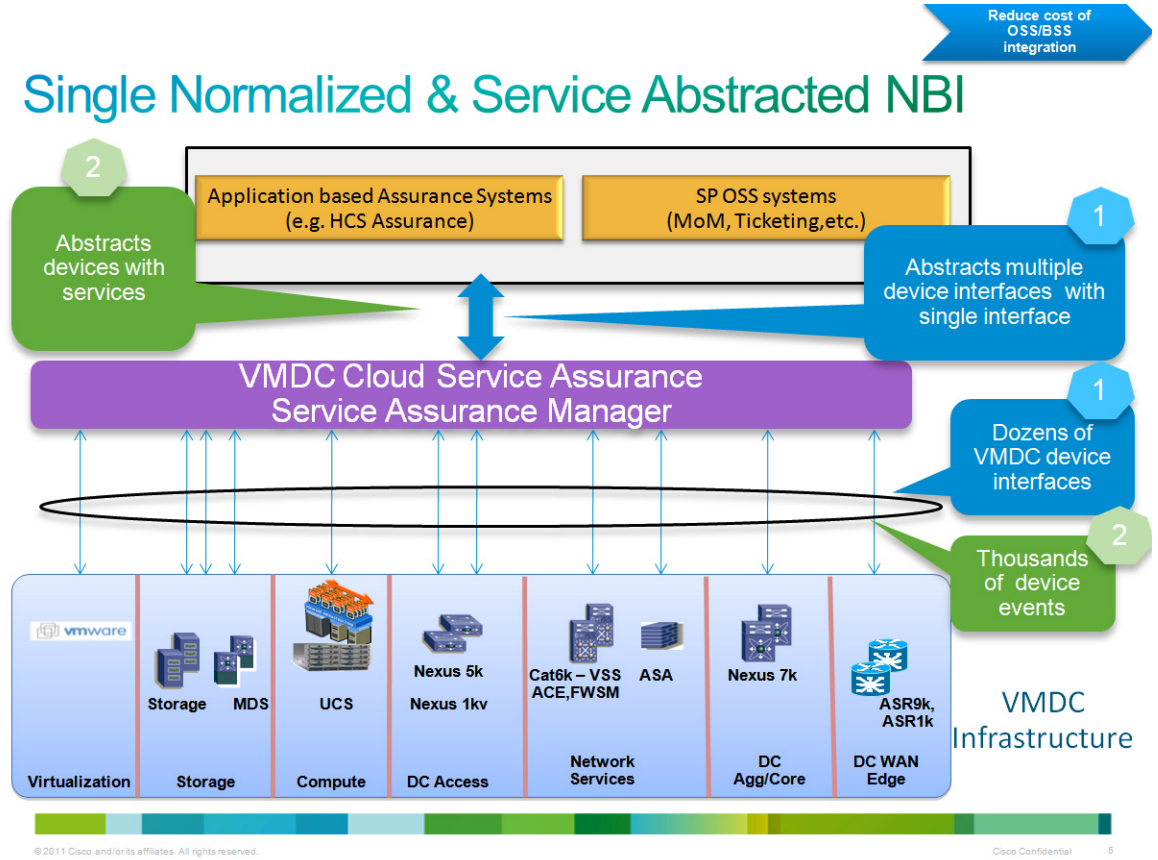
### Abstraction via Single Interface

One of the key functions of the SAM layer as defined in CLSA-VMDC architecture is the capability to provide a single, normalized NBI that is consistent regardless of the formats of data used by the underlying VMDC components. This allows simplified integration and ongoing interface maintenance with providers existing OSS systems as:

- There is only one integration point as opposed to the number of integration points being proportional to the number of VMDC devices and domain managers.
- Changes in any of the underlying interfaces on managed devices are absorbed by the SAM as opposed to the provider having to update OSS systems every time there is a change in one of the managed components.

Figure 4-11 illustrates how the VMDC system is abstracted via a single interface to the provider's existing OSS system.

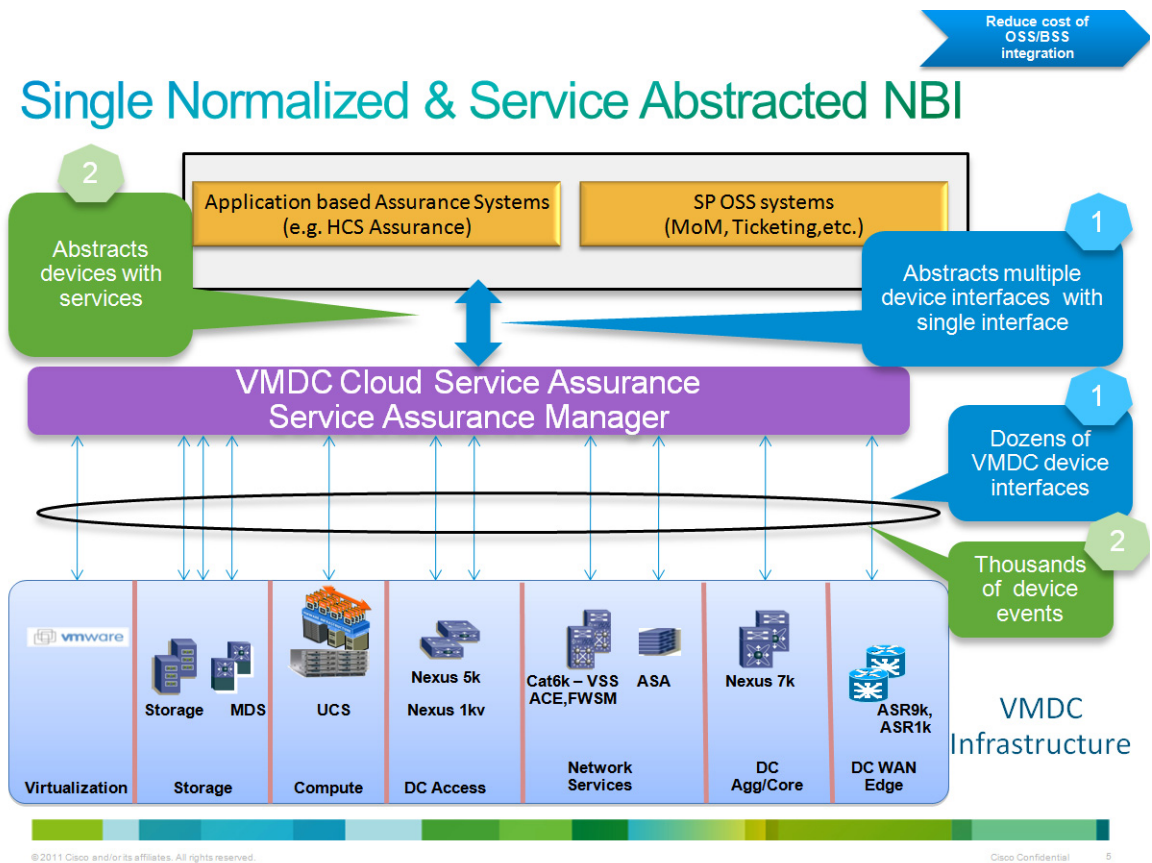
Figure 4-11 Single Normalized and Service Abstraction NBI



### Integration With Multiple Northbound Systems

This use case example illustrates the need for different types of notifications and northbound filtering capabilities.

Figure 4-12 VMDC CSA Integration in Northbound OSS Systems



In this use case example, there is an IT department with two operations teams: one for network operations and one for server/compute operations. In addition, each one of the teams has a ticketing system and a MoM capable of further event processing and RCA. Assume also that the server/compute operations team has an SLA management system used to measure and manage SLA compliance.

Using the extensive filtering capabilities of the northbound notifications, the needs of both of these operations teams and their various northbound systems can be satisfied with a single instance of the service assurance system. In this example, five northbound notification destinations are configured, each with a different filter (also known as a notification trigger) as follows:

- All root cause events originated by vCenter or UCSM are sent to the Compute Operations ticketing system.
- All service-impact events originated by vCenter or UCSM are sent to the Compute Operations SLA management system.
- All other compute events that may require additional analysis are sent to the Compute Operations MoM.
- All root cause events originated by network devices are sent to the Network Operations ticketing system.
- All other compute events that may require additional analysis are sent to the Network Operations MoM.

### Abstraction Through Service Overlays

This use case illustrates the need for service impact notifications from CLSA-VMDC. This use case is a prerequisite for integrating CLSA-VMDC into CLSA-HCS. To deliver HCS services (voice, voicemail, etc.) to the end customer/tenant, multiple services need to be provided to the customer, which are referred to as service overlays. In a scenario for top-level service such as HCS, there are a number of benefits to only processing abstracted events related to a few underlying services:

- Complexity of its fault management system can be reduced significantly if it is only receiving events related to few underlying services (IaaS, MPLS VPN WAN service, etc.) rather than having to deal with device level events from tens of underlying components.
- More flexibility to support various business and operational deployment models that vary in which domains and services are owned and operated by the provider offering top level (e.g., HCS) services.

Figure 4-13 and Figure 4-14 illustrate the service overlay approach for application-based services such as HCS, and the need for service level abstraction from the underlying infrastructure system.

Figure 4-13 HCS Services and Operational Domains

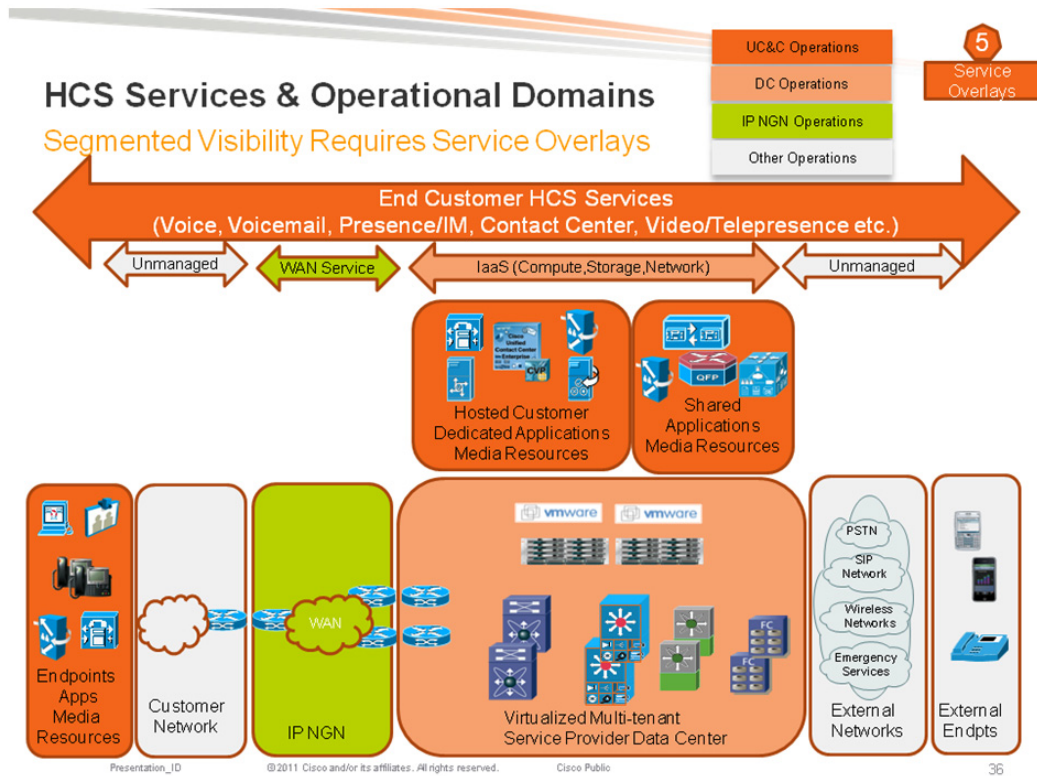


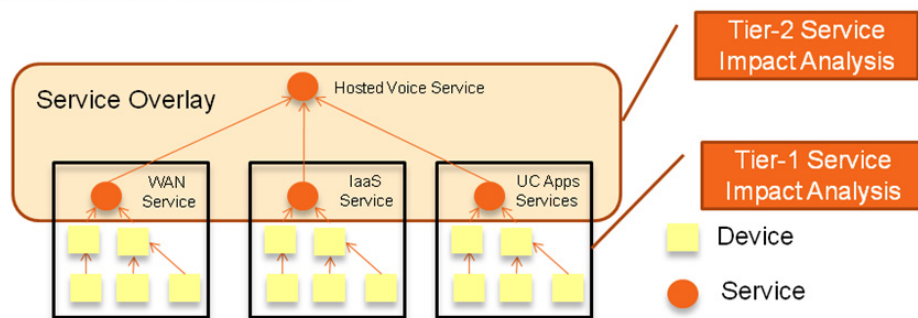
Figure 4-14 Multi-tiered SIA

## Multi-tiered service-impact analysis

Service Overlays: Analyzing services not device data

5  
Service  
Overlays

- End user experience and E2E SLAs depend on the full traffic path traversing multiple service and operational domains
- Define and monitor service events & service KPI/KQIs
- Simplify X-domain RCA/SIA by analyzing/correlating service level data rather than device level data



Presentation\_ID

©2011 Cisco and/or its affiliates. All rights reserved.

Cisco Public

37

## Performance Management

The following performance management capabilities are provided out-of-the-box in CLSA-VMDC:

- Key Key Performance Indicators (KPIs) resource monitoring and trending:
  - List of KPIs supported out-of-the-box for VMDC devices are listed in Appendix # Key Performance Indicators for VMDC.
  - Resource monitoring is partially validated as part of CLSA-VMDC.
- Performance service impact models for compute and storage:
  - TCA's utilized as part of SIA
  - Validated as part of CLSA-VMDC
- Application Response Time measurements:
  - Not validated as part of CLSA-VMDC
  - For details, refer to product documentation on [www.zenoss.com](http://www.zenoss.com).
- Performance Reporting:
  - Not validated as part of CLSA-VMDC
  - For details, refer to product documentation on [www.zenoss.com](http://www.zenoss.com).

## Dashboards

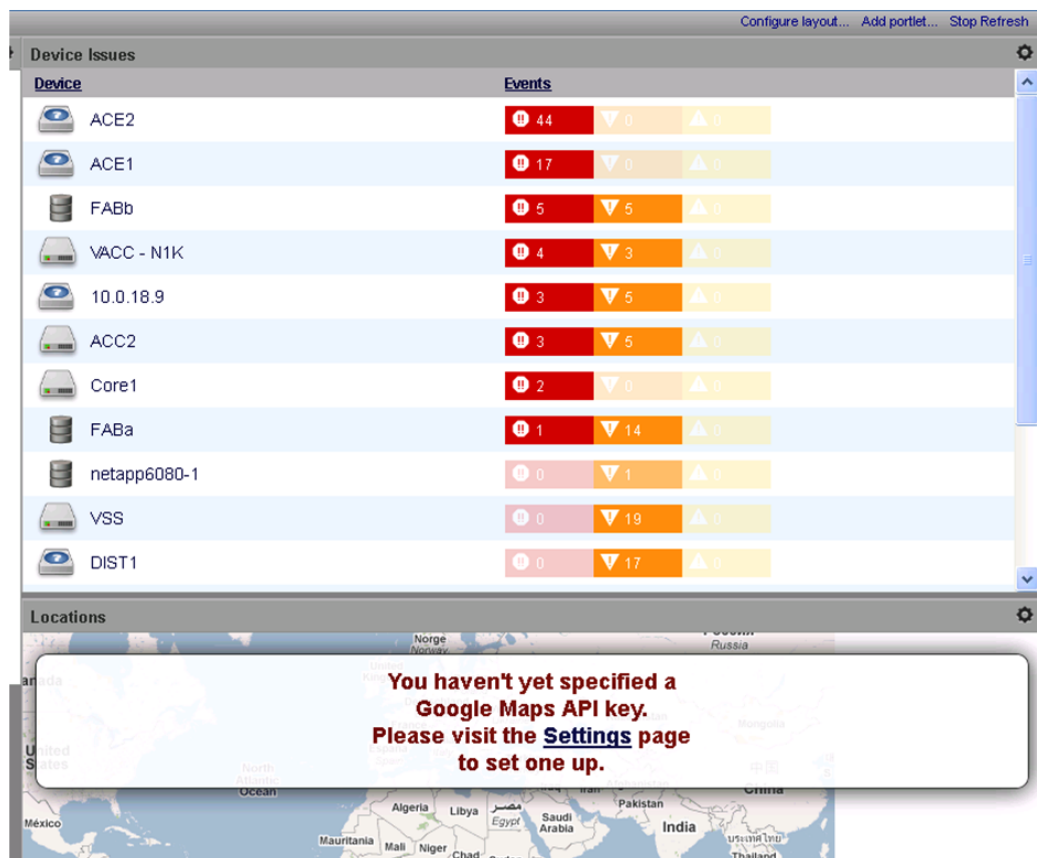
CLSA-VMDC features aggregated SP dashboards, as well as both device level and service level dashboards that operators can use to obtain more details. The following are the key dashboard categories for Zenoss Service Dynamics:

- [Aggregated Systemwide Resources Status Dashboards](#), page 4-29
- [Service Inventory and Status Dashboards](#), page 4-30
- [Infrastructure/Resource Monitoring Dashboards](#), page 4-31
- [Event Dashboards](#), page 4-33

### Aggregated Systemwide Resources Status Dashboards

These dashboards list all devices with events systemwide, sorted by number of highest priority events (Figure 4-15).

**Figure 4-15** *Aggregated Systemwide Resources Status Dashboard*



### Service Inventory and Status Dashboards

These dashboards show the availability and performance state of all services in the system (Figure 4-16).

Figure 4-16 Service Inventory and Status Dashboard

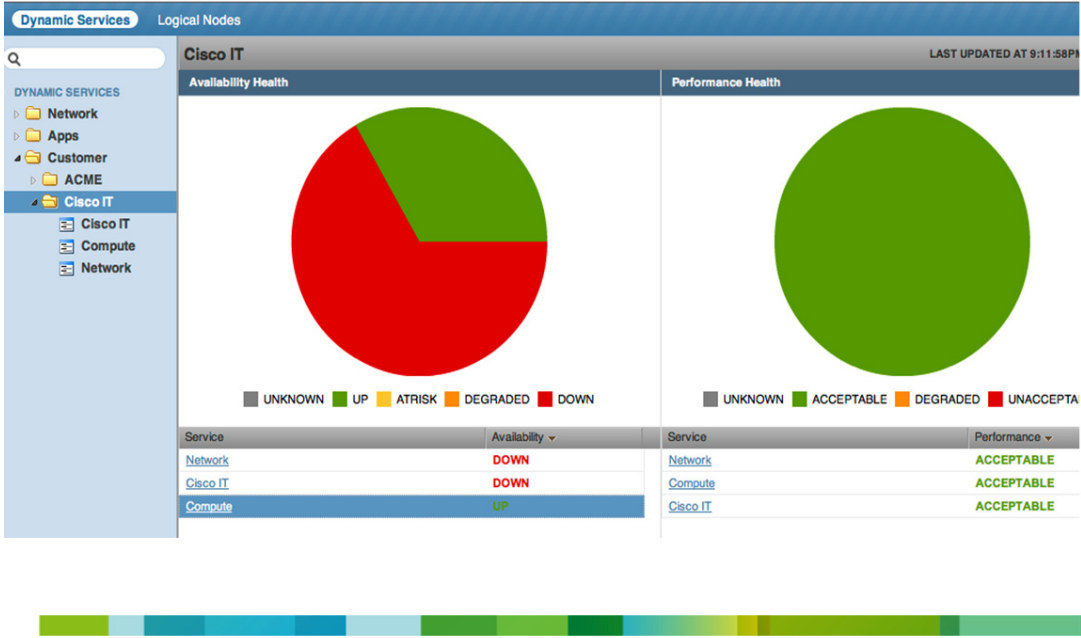
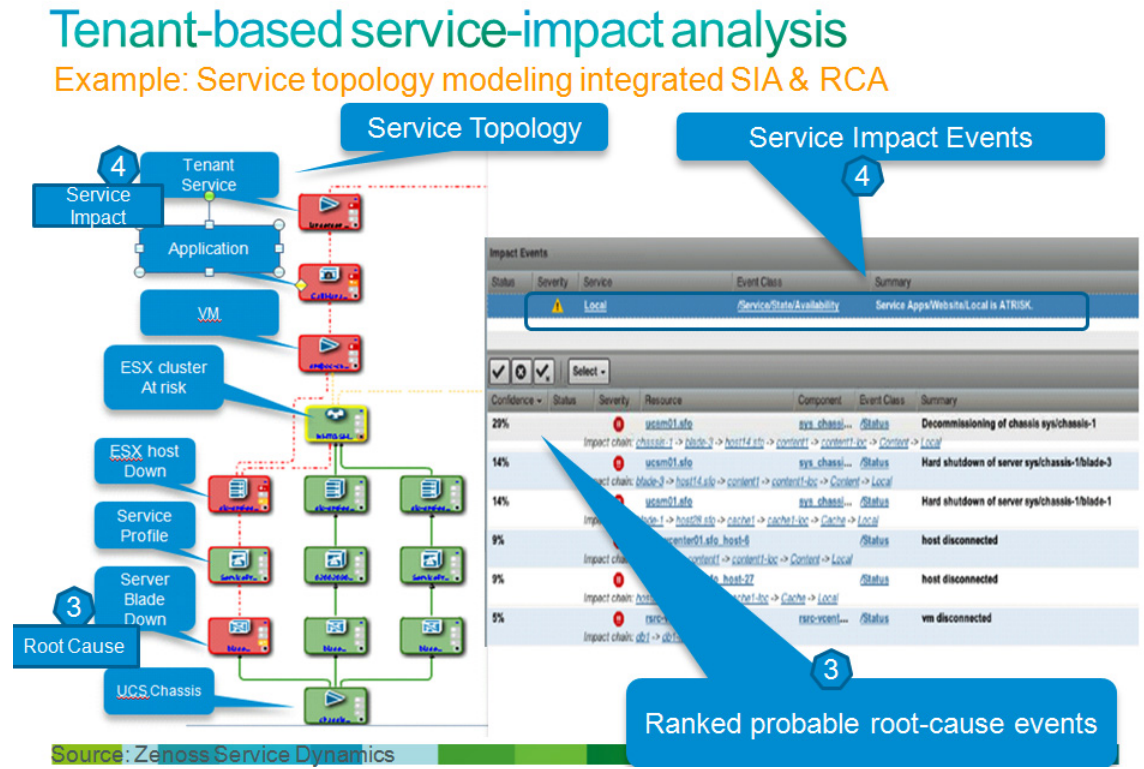


Figure 4-17 shows a per-service detailed dashboard, which lists service impact events and related probable root cause events, as well as a visualization of the service model tree.

Figure 4-17 Per-Service Detailed Dashboard



### Infrastructure/Resource Monitoring Dashboards

These dashboards list the inventory of all devices and their status (Figure 4-18).

Figure 4-18 Infrastructure Dashboard

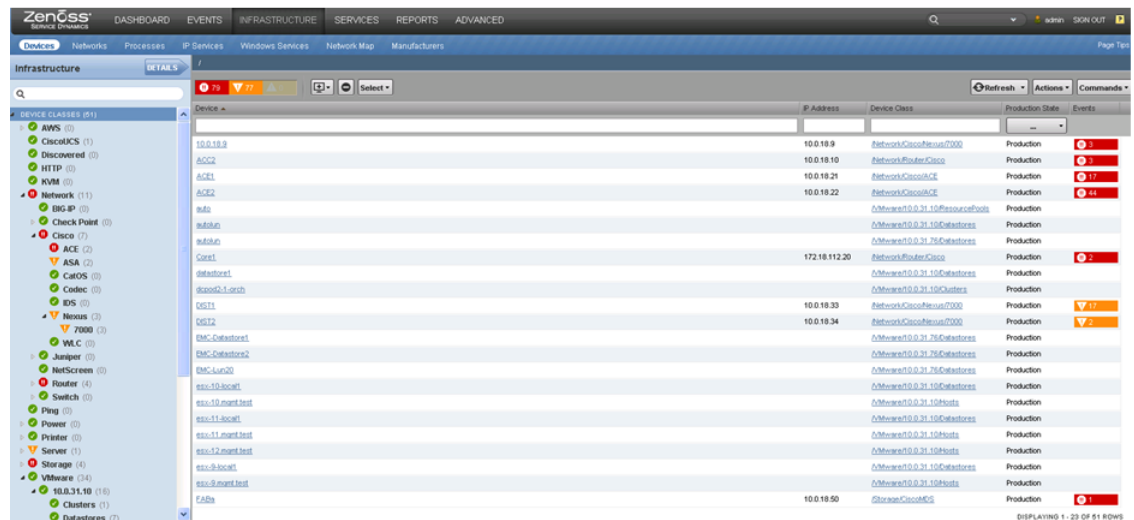


Figure 4-19 and Figure 4-20 show a detailed component dashboard and graphical view (example UCS server blade).

Figure 4-19 Detailed Component Dashboard

# Zenoss Cloud Service Assurance

## Fault Monitoring of VMDC Components

### Availability and Fault Monitoring

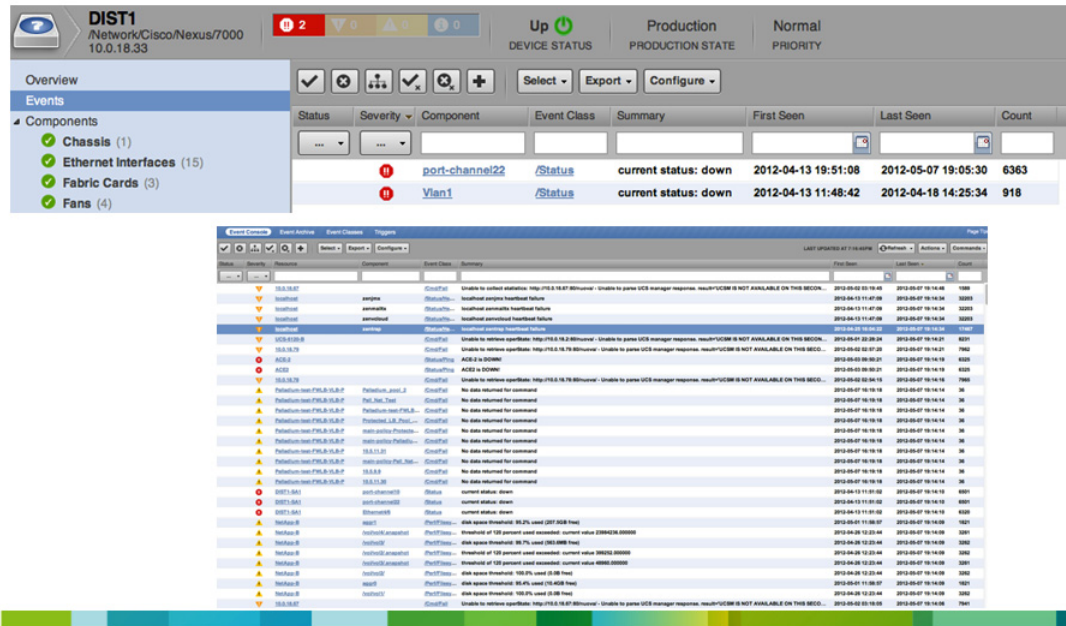
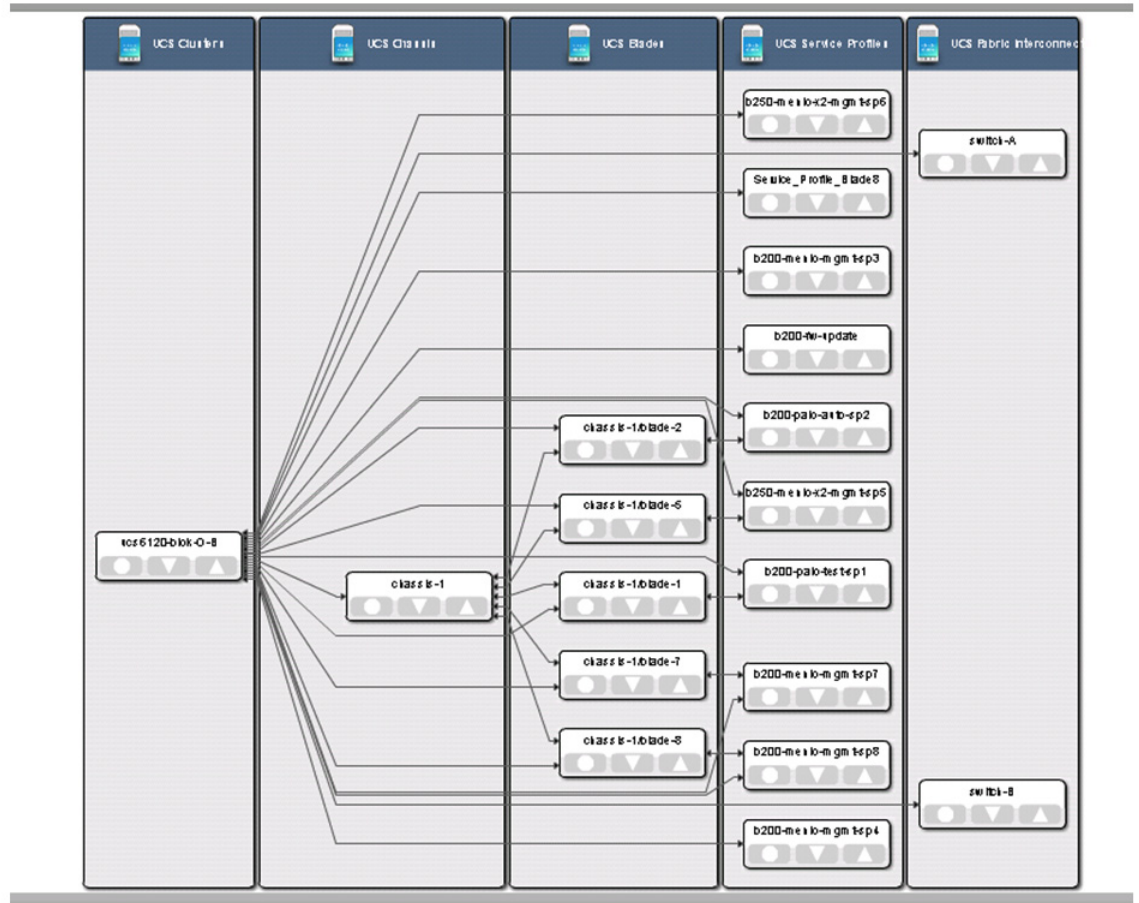


Figure 4-20 UCS Server Blade Graphical View

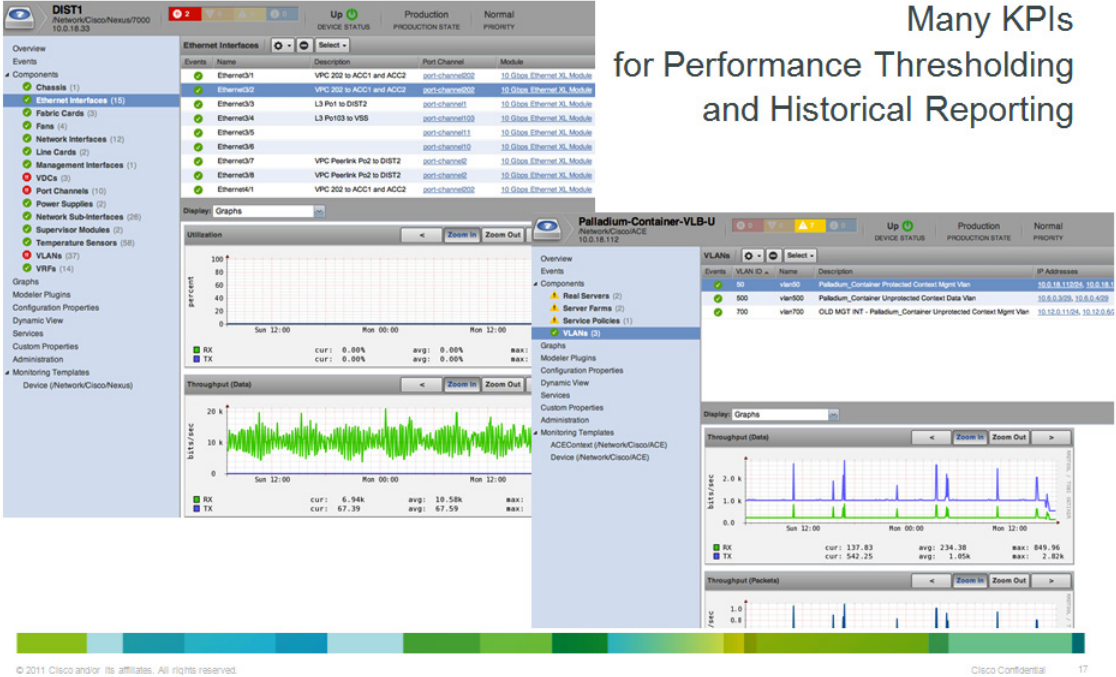


## Event Dashboards

These dashboards show all events in the console (similar consoles exist per component as well) (Figure 4-21).

Figure 4-21 Event Dashboard

# Zenoss Cloud Service Assurance Performance Monitoring of VMDC Components



Many KPIs for Performance Thresholding and Historical Reporting

## Reporting

CLSA-VMDC provides a range of defined and custom report options, including the following:

- Device reports
- Event reports
- Performance reports
- Graph reports
- Multi-graph reports
- Custom device reports

Reports can be exported to external files and systems or can be viewed locally. Reports can also be generated ad hoc or scheduled. See the [Zenoss Resource Manager Administration Guide](#) for more information.

The following is a list of reports supported out-of-the-box for CLSA-VMDC:

- Device Reports (9)
  - All Devices
  - All Monitored Components
  - Device Changes

- MAC Addresses
  - Model Collection Age
  - New Devices
  - Ping Status Issues
  - SNMP Status Issues
  - Software Inventory
- Custom Device Reports
- Graph Reports
- Multi-graph Reports
- Event Reports (3)
  - All EventClasses
  - All EventMappings
  - All Heartbeats
- Performance Reports (7)
  - Aggregate Reports
  - Availability Report
  - CPU Utilization
  - Filesystem Util Report
  - Interface Utilization
  - Memory Utilization
  - Threshold Summary
- Storage (3)
  - Clients
  - Licenses
  - Disk Firmware
- Enterprise Reports (17)
  - Organizer Graphs
  - 95th Percentile
  - Defined Thresholds
  - Interface Volume
  - Network Topology
  - Customized Performance Templates
  - User Event Activity
  - Notifications and Triggers by Recipient
  - Datapoints by Collector
  - Organizer Availability
  - Maintenance Windows
  - Interface Utilization

- Event Time to Resolution
- Data Sources in Use
- Users Group Membership
- Cisco Inventory
- Guest to Datapools
- MExchange (1)
  - MExchangeAvailability
- VMware (5)
  - ESXs
  - VMware Utilization
  - VMs
  - Datastores
  - Clusters
- Cisco UCS Reports (2)
  - Hardware Inventory
  - Free Slots

## Multi-Tenancy

This section discusses the CLSA-VMDC approach to multi-tenancy. VMDC architecture supports multi-tenant service delivery, and CLSA-VMDC must therefore support an assurance window into these tenant services to equip cloud providers with the ability to assure logically distinct customer services. A related topic, Role-Based Access Control (RBAC), is also presented.

### CLSA-VMDC Multi-tenancy

VMDC provides a multi-tenant services cloud infrastructure by logically separating tenant services that are implemented on a shared physical infrastructure. Tenants consume a portion of network, storage, and compute resources that have been allocated from the larger pool represented by the cloud. CLSA-VMDC delivers cloud provider assurance of shared infrastructure devices and their sub-components. In addition, CLSA-VMDC supports the multi-tenant services aspect of the VMDC architecture through the use of defined Tenant Services.

Zenoss CSA enables an administrator to stitch together service element nodes which taken as a whole comprise a specific tenant service. A CLSA-VMDC tenant service begins with creation of the topmost element node named for the tenant. To this tenant node, underlying VMware vSphere and UCS shared infrastructure elements can be discovered and attached. See [“VMDC Assurance Service Models” section on page 4-17](#) for more details regarding tenant services.

Using the tenant service modeling feature of CLSA-VMDC, cloud customers' services can be assured independently. Elements of the tenant service that are unique to that tenant customer such as specific VM's are visible only to the cloud provider or the service owner. Elements of the service that belong to shared infrastructure, such as a UCS chassis or a storage device are visible across multiple tenant services, as would be expected. In fact, if a shared device experiences a fault condition, all services associated with that device should be impacted. However, any fault condition associated with unique elements of a tenant service are not visible to other tenants.

**Note**

This phase of CLSA-VMDC only supports cloud provider visibility into dashboards and service impact trees. Tenant customer visibility into service impact trees via customer portals will be supported in future releases.

## RBAC Implementation

As a cloud providers' infrastructure increases in scale, it becomes important to provide a segmentation of operations capability to implement a division of responsibility. CLSA-VMDC fulfills this need with its RBAC implementation. Beyond this division of responsibility capability, RBAC can also be used to support groups of users with limited visibility into specific tenant services. [Table 4-5](#) lists the out-of-the-box roles that may be used to segment cloud provider operations responsibilities and access.

**Table 4-5 Global User Role Definitions**

Role	Definition
ZenUser	Provides global read-only access to system objects.
ZenManager	Provides global read-write access to system objects.
Manager	Provides global read-write access to system objects and read-write access to the Zope object database (which includes all devices, users, and event mappings)
ZenOperator	Provides users the ability to manage events, i.e. acknowledge, move to archive, etc.

These predefined roles are global in scope, such that an operator may access all cloud objects, but only be allowed certain operations. In addition to these global user roles, users may be defined that are more limited in scope. These user roles may be assigned to organizational groups to manage a subset of the entire infrastructure or even specific tenant services.

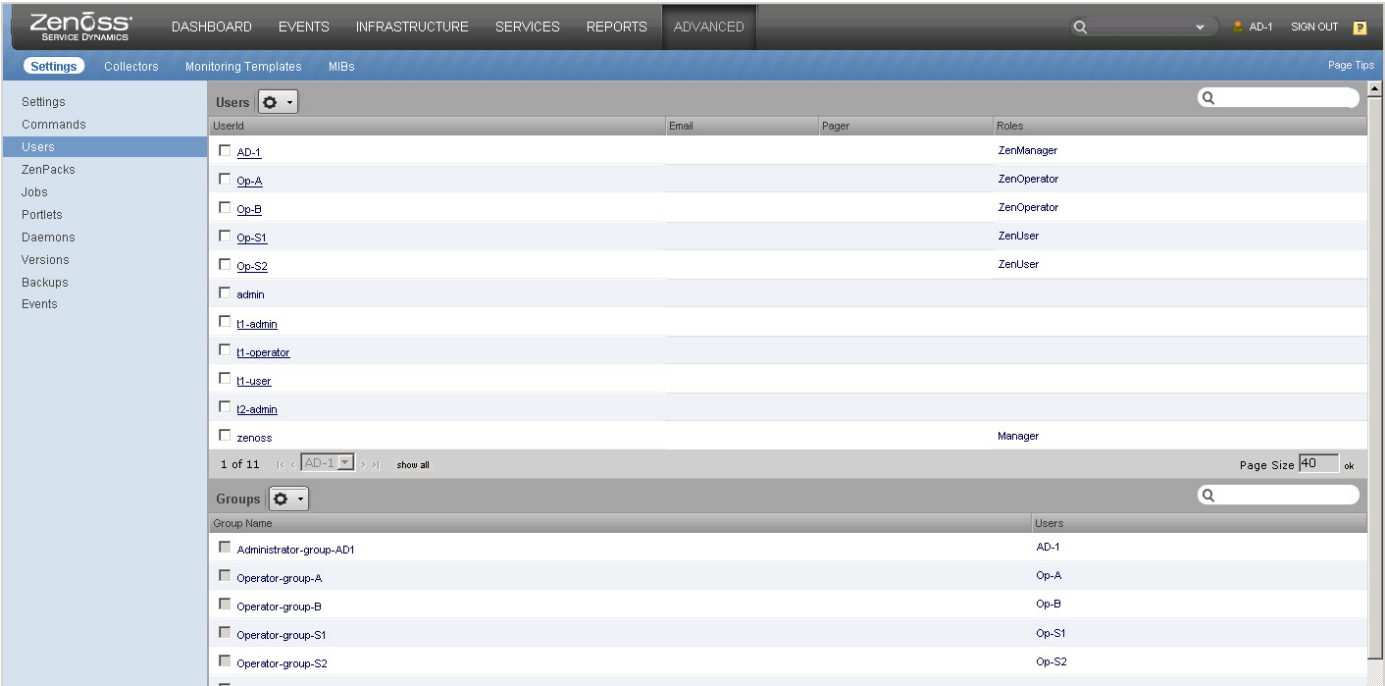
Organizational groups are used to collect subsets of infrastructure and/or services into logical categories for segmented operations. [Table 4-6](#) lists the broad group categories and suggested uses for each group type. Groups can be devised for each tenant such that a customers tenant services can be assigned to a tenant group.

**Table 4-6 Device and Service Group Categories**

Group Categories	Group Purpose
Group	Can be used for collecting similar devices, e.g. all switches group, all compute devices group
Systems	Can be used to collect all equipment with a specific data center, e.g. data center A, data center B
Locations	Can be used to collect devices by geographic boundaries, e.g. city, state, or even specific device rack

[Figure 4-22](#) illustrates a list of users with either global or customized group role assignments. Users with the global user roles would belong to the cloud provider.

Figure 4-22 Custom User Groups



Users with non-global roles can belong to either the cloud provider or even a cloud commuter. While RBAC is only supported for the cloud provider in this phase, it will be the mechanism to deliver a cloud customer, or multi-tenant services, assurance portal in future phases.