Turbonomic 5.7
REST API Guide

Turbonomic, Inc.

500 Boylston St, 8th floor
Boston, MA 02116 USA
Phone: (866) 634 5087
www.turbonomic.com

COPYRIGHT

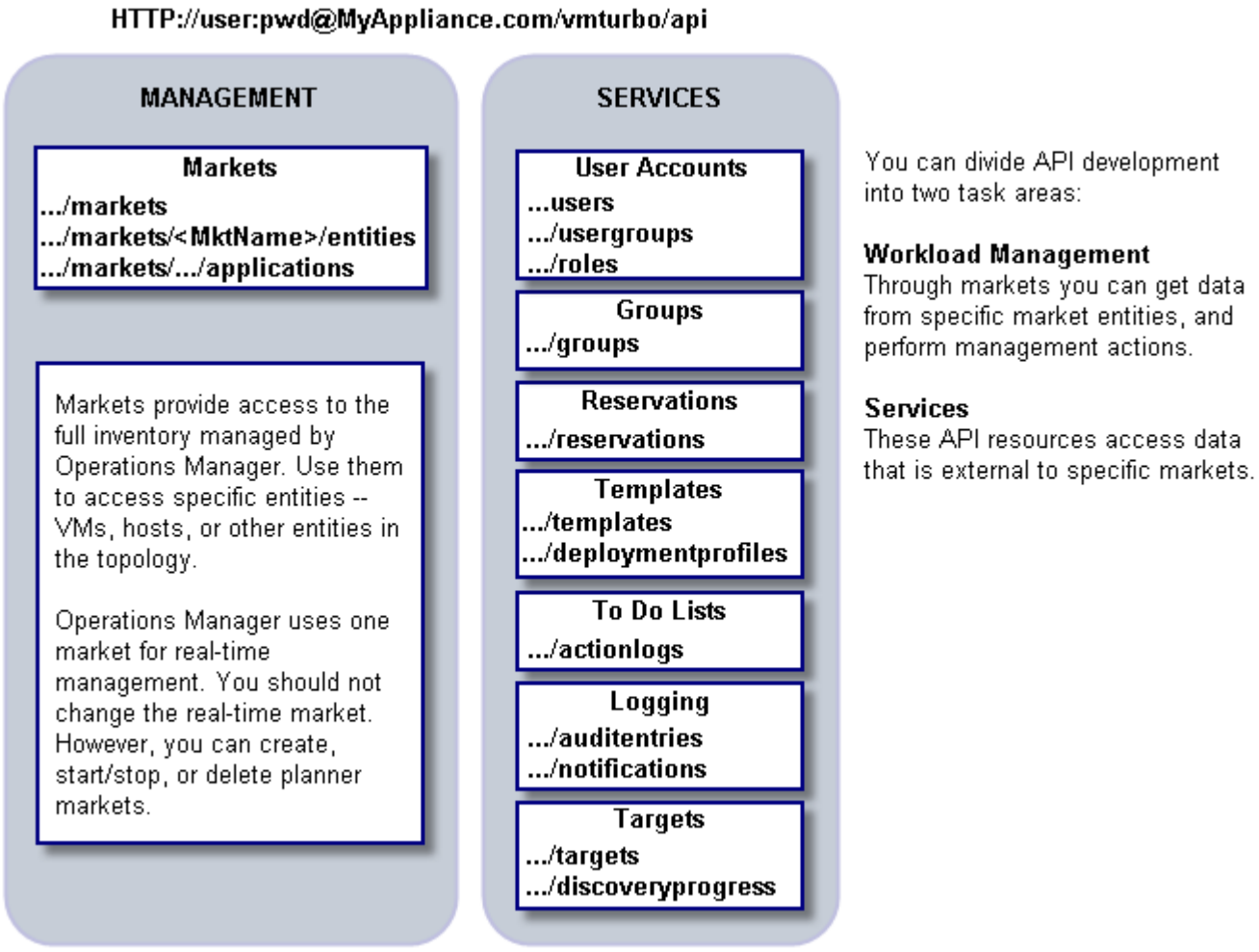
END-USER LICENSE AGREEMENT

https://cdn.turbonomic.com/wp-content/uploads/Turbonomic_Click_Through_Customer-License.pdf

Table of Contents

Data Model

The REST API exposes Turbonomic data and processing to remote access via HTTP GET, POST, and DELETE methods. To get the most out of the REST API, you should understand how Turbonomic organizes its underlying data, and how the various REST resources map to that organization.



This manual describes markets and the service resources in detail, along with the methods you can call to use them.

For even more information, you can see the following Blog Series on the Green Circle: Green Circle VMTurbo REST API Series.

Accessing Markets

Turbonomic uses market-based analysis to perform workload management. To do this, it constructs a model of your environment, representing each entity as a buyer and seller in a market. You can access this model via a named market resource.

At any time, your Turbonomic appliance can have a number of markets in memory. To get a list of the current markets in your appliance, execute the following URL:

`https://<username:password>@<appliance>/vmturbo/api/markets`

This listing will include at least two markets (the template market and the real-time market).

NOTE: When you get a list of action logs (To Do lists), each entry includes the name of its associated market in its `displayName` and `name` attributes. In this way, you can use market names to identify whether the To Do list is for the real-time market, or for a given plan.

Market Types

The market name indicates the following market types:

- Template market
- Real-time market
- Internal-use plans
- User-created plans

The Template Market

The template market is named `Market_Default`. This market should always be in the `STOPPED` state. You should never access this market for any reason.

The Real-Time Market

The real-time market is named `Market`. This market should always be in the `RUNNING` state.

The real-time market performs analysis and workload management on your environment. Turbonomic performs discovery to populate the topology it manages (the collection of entities in the real-time market). For this reason, you should be careful not to delete entities in the real-time market. The only POST and DELETE methods you should perform on the real-time market are to manage external applications.

NOTE: It is possible to execute POST or DELETE methods to modify the real-time market. However, it is highly unlikely that you would have a reason to do so. You should be aware that changes to the real-time market will affect the analysis Turbonomic performs. For example, if you remove a VM from the real-time market, Turbonomic can no longer manage its placement. The VM will still be present in your environment, but it will no longer be managed by Turbonomic. However, you can't reliably use this technique to make specific entities unavailable to Turbonomic. For the next discovery pass, Turbonomic will rediscover the deleted entity, and it will appear in the real-time market again.

You can use the real-time market to access entities and get current or historical data about them.

Planner Markets

Your appliance can have markets other than the real-time market (`Market`) and the template market (`Market_Default`). The appliance uses these other markets to represent plans.

A plan is actually a snapshot of the real-time market. Turbonomic runs economic cycles (buy/sell cycles) against this snapshot market until there are no more meaningful improvements to be made. At this point the plan run is completed.

Internal-Use Plans

Turbonomic runs plans to generate data that it displays in the GUI — The Cluster Capacity and Project Cluster Resources dashboards both display data generated by regularly-run plans. You should not make any changes to these markets.

A plan is for internal use includes the substring `_BasePlan` in the market name — You should not modify these plans.

User-Created Plans

Users can create plans to run what-if scenarios in the environment. At any time, each user account can have a planner market loaded in the appliance. This means that the number of resident user-created plans can potentially be one for each user logged into the appliance.

You identify planner markets by their names. A plan name is specified as `<user_uuid>_BasePlan<Plan_Type>`.

For example, a valid plan name is `_2cQ6kHszEeGmM4v_fWCzJw_BasePlanProjection`. To find out which user owns this plan, you can parse out the user's uuid and query the API for that user's account information.

A planner market can be based on a plan file that is saved on the appliance. However, the planning market name does not correspond with the name of the plan file.

The REST API includes a number of POST and DELETE calls for markets. You will almost always make these calls to user-created planner markets, and almost never to the real-time market. The actions you can perform on a plan's market include:

- Create/Delete a plan
- Run or stop a plan
- Set plan scope
- Modify the plan's workload (for example, add VMs to a plan or delete entities from a plan)
- Add or delete placement policies in a plan
- Delete access commodities from a plan (remove placement constraints)

Plan Scope

Plans can have a scope set to them. When you create a plan, you can provide an array of groups to set the plan's scope. To change a plan's scope, you must delete the plan and create a new one with a different scope. You specify scope by passing an array of group entities in the call to create the plan.

Plan State

Planner markets can be in one of the following states:

- `RUNNING`
- `STOPPED`
- `READY_TO_START`

If a planner market is running, then the appliance is calculating the plan results. If the market is stopped, then the plan has been run and you can access data from this market to see the target results for the plan. You can make a POST call to change the plan's state — for example, you can stop a running plan.

NOTE: It's possible to stop a planner market through the GUI or the API. Also, plan may have been stopped before it finished its calculations. In this case, the plan results will be incomplete.

Using the POST Method with Markets

POST requests modify the market. You should never send a POST request to the template market (named `Market_Default`). You should be careful when sending a POST request to the real-time market (named `Market`) — typically, you will only modify the real-time market to define applications on it or to start a deployment action.

Most of the market modifications you perform will be on a planner market. You identify planner markets by their names. A plan name is specified as `<user_uuid>_<planType>`. Before you modify a plan, you should check the plan name against the list of users to make sure you are modifying the plan you want.

NOTE: To make POST requests on a market, the URL login credentials must be for an account with an `advisor`, `automator`, or `administrator` role.

You can perform the following POST actions in a market:

- Create or modify a plan
  - Create a plan (if the named market doesn't exist)
  - Run/stop a plan
  - Enable/disable a plan
  - Set plan scope
  - Add clones of templates to the plan

  For more information, see POST: markets — creating or modifying a plan on page 64.
- Add entities to a plan
  See POST: entities — add entity copies to plan on page 88.
- Specify placement policies in a plan
  See POST: entities — specify placement policies in a plan on page 90.

Using the DELETE Method with Markets

DELETE requests remove the market from Turbonomic memory, or they remove specified entities from the market. You should never send a DELETE request to the real-time market (named `Market`) or the template market (named `Market_Default`).

When deleting a market, you should only delete a planner market. You identify planner markets by their names. A plan name is specified as `<user_uuid>_<planType>`. Note that many users can have plans loaded in the appliance. Before you modify a plan, you should check the plan name against the list of users to make sure you are modifying the plan you want.

> **NOTE:** To make DELETE requests on a market, the URL login credentials must be for an account with an `advisor`, `automator`, or `administrator` role.

If you have added copies of entities to a plan, you can also remove those copies from the plan (see DELETE: entities — delete entity from a plan by type on page 91). A plan can include access commodities to specify workload placement constraints — you can also delete these from a plan (see DELETE: entities — delete access commodities from a plan on page 92).

Accessing Market Entities

Each market manages a set of entities — using a market's name you can get access to the entities in that market. As you access market entities, you should keep the following in mind:

- Entities in the real-time market reflect the current state in your physical environment
- You should not add or remove entities in the real-time market
- Entities in a plan market reflect a snapshot of the environment from when the market was created
- Plan markets can be scoped to a subset of your physical environment

For a given market, you can then access the following types of entities:

- Virtual applications
- Applications
- VMs
- Hosts
- Datastores
- Disk arrays
- Storage controllers
- Switches
- Virtual datacenters
- Datacenters

For example, the following URL will get all the datastores in the real-time market:

```
https://administrator:administrator@10.10.172.62/vmturbo/api/markets/Market/
entities?classname=Storage
```

For each entity type, you can:

- Get all instances of that type
- Get instances by name
- Get historical data for an instance
- Get commodity information for an existing instance
    - List of commodities
    - Commodity details
    - Commodity Bought details
- Get actions action or notification information related to an existing instance
    - Related actions
    - Related risks/opportunities (notifications)
    - Related audit log entries
- Clone entities into a plan market from templates or existing instances
- Replace an existing instance in a plan market with a template clone
- Create or delete placement policies by adding or removing access commodities.
- Delete existing instances from a plan market

For example, the following URL queries the real-time market for all the VMs that are hosted by a PM named `host-59`:

```
https://administrator:administrator@10.10.172.62/vmturbo/api/markets/Market/hosts/host-59/virtualmachines
```

Entity Resources and Services

In the market each entity is modeled as a buyer and a seller. In the API, the capacity that an entity buys is represented by `resources`, and the capacity it sells is represented by `services`.

For example, the following URL lists the resources that are bought by the PM named `host-59`. For a PM, these resources can include `StoageAccess`, `StorageLatency`, `Space`, `Power`, or `Cooling`. The returned list includes the names of the providers of these resources:

```
https://administrator:administrator@10.10.172.62/vmturbo/api/markets/Market/hosts/host-59/resources
```

In this example, the following URL lists the services that are sold by the PM named `host-59`. For a PM, these services can include CPU, `Mem`, `NetThroughput`, and so on. In addition, a PM can sell commodities that identify placement, such as `ClusterCommidity` to identify the cluster the PM belongs to, or `DatastoreCommidity` items to identify the datastores that support the PM:

```
https://administrator:administrator@10.10.172.62/vmturbo/api/markets/Market/hosts/host-59/services
```

Access Commodities Specifying Entity Placement

Turbonomic includes Workload Placement Segments, which are policy settings to constrain workload placement. These segments identify providers, and determine which consumers can purchase commodities from those providers.

To specify workload placement via the API, use entity resources and services to create access commodities. An access commodity specifies the `services` that an entity provides, or the `resources` an entity consumes. To establish workload placement, you set up a bi-directional relationship between service providers and resource consumers.

For example, assume you create `services` access commodities for Host_A and Host_B. For each host, you specify a different set of datastore services the host will provide. Then for a given VM, you can specify a `resources` access commodity to specify that the VM purchases resources from either Host_A or Host_B. The choice of host determines which datastore resources the VM can use.

Other examples include:

- Limit a set of VMs to a single host:
  Specify `services` access commodities for that host to provide the necessary services, then specify `resources` access commodities for each VM to consume from that host.
- Limit a set of VMs to two specific hosts, but one datastore:
  Specify two `services` commodities for the given datastore to provide datastore services to both hosts. Then specify `resources` access commodities for each VM to consume from that datastore.

Using the POST Method with Entities

POST requests modify the market. You should never send a POST request to the template market (named `Market_Default`). You should be careful when sending a POST request to the real-time market (named `Market`) — typically, you will only modify the real-time market to define external applications on it.

Most of the market modifications you perform will be on a user-created planner market. You identify planner markets by their names. A plan name is specified as `<user_uuid>_<planType>`, where `planType` is one of:

- `BasePlan` for Workload Distribution
- `BasePlanProjection` for Workload Projection
- `BasePlanHWReplace` for Hardware Replacement

For example, a valid Workload Distribution plan name is `_2cQ6kHszEeGmM4v_fWCzJw_BasePlan`. Note that many users can have plans loaded in the appliance. Before you modify a plan, you should check the plan name against the list of users to make sure you are modifying the plan you want.

For more information about the different types of markets, see Market Types on page 8.

NOTE: To make POST requests on a market, the URL login credentials must be for an account with an `advisor`, `automator`, or `administrator` role.

You can make POST requests for entities to make the following changes on a planner market:

- Add entities to a plan
  See POST: entities — add entity copies to plan on page 88.
- Specify placement policies in a plan
  See POST: entities — specify placement policies in a plan on page 90.

Using the DELETE Method with Entities in a Plan Market

DELETE requests remove specified entities from a plan market. You should never send a DELETE request to the real-time market (named `Market`) or the template market (named `Market_Default`).

You identify planner markets by their names. A plan name is specified as `<user_uuid>_<planType>`, where `planType` is one of:

- `BasePlan` for Workload Distribution
- `BasePlanProjection` for Workload Projection
- `BasePlanHWReplace` for Hardware Replacement

For example, a valid Workload Distribution plan name is `_2cQ6kHszEeGmM4v_fWCzJw_BasePlan`. Note that many users can have plans loaded in the appliance. Before you modify a plan, you should check the plan name against the list of users to make sure you are modifying the plan you want.

NOTE: To make DELETE requests on a market, the URL login credentials must be for an account with an `advisor`, `automator`, or `administrator` role.

You can make POST requests for entities to make the following changes on a planner market:

- Remove entities from a plan
  See DELETE: entities — delete entity from a plan by type on page 91.
- Remove workload placement policies from a plan
  See DELETE: entities — delete access commodities from a plan on page 92.

External Applications in the Market

By default Turbonomic discovers applications that are running in your environment (see Application Discovery in the Turbonomic Policies view). However, the REST API provides the `applications` resource that you can use to assign external applications to a VM or a group of VMs.

When working with this resource, you can create two types of application:

- Applications that you discover with processes external to Turbonomic
- Jobs that you set up to be executed by the Turbonomic Actions Manager

Externally Discovered Applications

The Turbonomic Application Control Module includes mechanisms to discover applications running in your environment. However, you might have some applications or processes with signatures that aren't recognized by the Application Control Module.

In that case, you can use your own external processes to discover and monitor applications in the environment. You can use the REST API to register these applications to appear in the Turbonomic GUI, but set their charted values (resource utilization and services provided) with data from your external process. In this way, you can integrate your external application monitoring with Turbonomic.

To work with these applications, you use POST methods to:

- Create external applications in the real-time market
- Set application state (external applications should always be `not_monitored`)
- Set application service values
- Set application resource values

When you create an application, the API returns an entry that includes the application's UUID. Given that, you can then set the application's state — for an externally discovered application, you should always set the state to `not_monitored`. This is because you will be performing the monitoring, and updating the services (what the application provides) and resources (what the application consumes).

As you create applications and update their services and resources, these changes will appear in the Turbonomic user interface.

Jobs Executed by the Actions Manager

Turbonomic includes an Actions Manager that controls the execution of externally defined jobs. For example, you might want to update the OS on all the VMs in a group. Executing such a job on a massive scale can flood your storage with IOPS consumption. The Actions Manager can limit the number of concurrent actions — by default it limits them to 400 per Storage Controller.

> **NOTE:** The current implementation of the Action Manager controls action execution on Storage Controllers. To use the Action Manager, you must have both the Storage Control Module and the VDI Control Module enabled on your appliance

To set up a managed action, you will:

- Script the jobs in the `START_Application.sh` file
  Each job in the file has a name to identify it. The job receives the name and UUID of the VM to run it on, as well as the name and UUID that Turbonomic uses to track the action.
- Create a matching application via the API
  You create the application for a VM or a group of VMs. The name you provide for the application must match a name that is in the `START_Application.sh` script. You also specify the number of action tokens this job will use. The Actions Manager has a capacity for the number of tokens that can be out at a time. Each action withdraws the number of tokens you specify.

The START_Applications.sh Script

This script contains the actions that you want to perform for each job — For example, upgrading the OS on every virtual machine, or running a security patch.

The following full path gives the required location and name for this script file:

`/srv/tomcat/script/control/START_Application.sh`

For every external application that invokes the action manager, this script will be called. A very simple sample script contains the following:

```
# UUID of Virtual Machine - $1
# Display Name of Virtual Machine - $2
# UUID of Job - $3
# Display Name of Job - $4
echo $1 ' : ' $2 ' : ' $3 ' : ' $4> ~/$2.txt
```

This script simply writes out text file with the name of the VM, and containing the values of the arguments passed into the script.

To script different jobs, you can test the $4 argument, and branch into the appropriate section.

When the job is finished, you must delete the associated application. You can use different approaches to do this:

- Manage applications externally
  From the caller or some other external utility, keep track of the UUIDs of the applications and delete them when you know the jobs are done.
- Delete from within the scripted job
  If your script calls the job synchronously, you can execute a call to delete the application as the last step of your job. Be sure to pass the UUID ($3) and display name ($4) of the application. The UUID is required to delete a single application, and the name is required to delete all the applications created for a group.

Creating an Application for the Job

To execute the job, you:

- Create an external application on the VM you want to modify
- Give the application a name that matches a job in your script
- Specify the number of action manager tokens this job consumes

You can create applications for individual VMs, or you can create applications for a group. When you create the application, the action manager will then recommend it in the To Do list, and call the `START_Applications.sh` script.

When the job is done, you must delete the application to free the consumed tokens. You can delete an application per VM or per group of VMs. To delete applications per VM, you must have the application's UUID. To delete applications per group, use the application name.

Service Calls

The Turbonomic server manages data that is globally available to all the markets. This data serves Turbonomic processing in different ways, and the API can get and set values through different service resources. This document divides these resources into the following categories:

- User Accounts
  Resources for user accounts, groups, and roles:
  - users: Specific user accounts, including user role and password.
  - usergroups: ActiveDirectory groups.
  - roles: The user roles currently supported on the appliance.
- Groups
  Resources that manage collections of entities:
  - groups: This includes groups that Turbonomic discovers, as well as groups that users create in the GUI or via the API.
  - deployitems: Listings of proposed VM deployments. Turbonomic generates these items in response to a reservation.
- Managed Reservations
  Resources for templates and deployment profiles:
  - templates: Discovered and user-authored templates for VMs, PMs, and storage.
  - deploymentrofiles: Discovered and user-authored deployment profiles. Use these with associated templates to make reservations and deploy VMs.
- Templates
  Resources for templates and deployment profiles:
  - templates: Discovered and user-authored templates for VMs, PMs, and storage.
  - deploymentrofiles: Discovered and user-authored deployment profiles. Use these with associated templates to make reservations and deploy VMs.
- Logging
  Resources that manage the different logs:
  - auditentries: This corresponds to the Turbonomic audit log — a listing of actions that have been performed on the entities in your environment.
  - notifications: This includes the Risks/Opportunities, and also notifications that identify network and configuration issues with Turbonomic (target loses connectivity, VMWare Tools not installed, etc.).
  - actionlogs: The To Do lists that are current for the appliance. Note that the name attribute of an action log includes the name of the market it is associated with. However, an action log is not a constituent member of a market.
  - inventory A summary of the entities in each target.
  - settings A listing of the action modes (Automate, Manual, Recommend, Disable) that are set to entities in each group.
- Targets
  Resources to manage targets and discovery:
  - targets: The target services that the appliance is managing. You can see the current targets, or configure new ones for the appliance.
  - discoveryprogress: A listing of the discovered entities on the appliance.

User Accounts

Turbonomic set up accounts to grant users specific access to the product. The API exposes user accounts through the following resources:

- users
  Get a list of all users, or get details about a specific user. Create a user, or delete a user.
- usergroups
  Specifications for Active Directory groups. A member of the specified group can log in using a valid User Principal Name.
- roles
  A list of the user roles your appliance supports, identified by UUID.

users

The following URL gets a listing of all the users on the appliance:

```
https://guest:guest@MyVmt.com/vmturbo/api/users
```

Each entry contains a `TopologyRelationship` element that identifies the user role by UUID. To get the name of the role, you can map this UUID to the items you get via the `roles` resource.

An entry can also contain or more `scope` elements that identify groups that are used to assign a scope to this user account. The scope element contains the UUID of a group — use that UUID to get the group's members via the `groups` resource.

To create a user you provide the following settings in a POST request to `users`:

- User name
- Password
- Role
  Provide the role as a string that matches the names of roles you get via the `roles` resource.
- User type
  Can be one of `DedicatedCustomer` or `SharedCustomer`.
- User scope
  An optional array of group UUID values, where the UUIDs identify groups of entities.

After you create the user account, you should see it in the list when you GET `users`.

To edit a user account, you make the POST request again, giving it the new settings that you want. You cannot use the API to incrementally change a user account. The new POST request overwrites the old account with new values.

Active Directory Groups

The `usergroups` resource manages Active Directory group accounts that are declared on the appliance. These accounts use Active Directory to manage authentication. You can role and scope for the group, and any member of the AD group can log into Turbonomic with those privileges.

The following URL gets a listing of all the AD user groups on the appliance:

```
https://guest:guest@MyVmt.com/vmturbo/api/usergroups
```

The returned data is very similar to data returned for user accounts from the `users` resource. Each entry contains a `TopologyRelationship` element that identifies the user role by UUID, and if the group is scoped it will contain one or more `scope` elements.

To create a user group you provide the following settings in a POST request to `usergroups`:

- Group name
- Role
  Provide the role as a string that matches the names of roles you get via the `roles` resource.
- Group type
  Can be one of `DedicatedCustomer` or `SharedCustomer`.
- Group scope
  An optional array of group UUID values, where the UUIDs identify groups of entities.

After you create the user group, you should see it in the list when you GET `usergroups`.

To edit a user group, you make the POST request again, giving it the new settings that you want. You cannot use the API to incrementally change a user group. The new POST request overwrites the old group with new values.

Roles

The following URL gets a listing of user roles that are supported on the appliance:

```
https://guest:guest@MyVmt.com/vmturbo/api/roles
```

The returned list contains the following elements:

```
<TopologyElement creationClassName="UserRole" displayName="Administrator Role"
name="administrator" uuid="_4UAioQY-Ed-WUKbEYSVIDw"/>
<TopologyElement creationClassName="UserRole" displayName="Observer Role"
name="observer" uuid="_4T_7lQY-Ed-WUKbEYSVIDw"/>
<TopologyElement creationClassName="UserRole" displayName="Advisor Role" name="advisor"
uuid="_4T_7lwY-Ed-WUKbEYSVIDw"/>
<TopologyElement creationClassName="UserRole" displayName="Automator Role"
name="automator" uuid="_4T_7mQY-Ed-WUKbEYSVIDw"/>
```

Each item has a `name` attribute that matches the value you give for a role when creating a user account. When you get a user account or a user group, the role for that account is included as a `TopologyRelationship` element. It includes the UUID of the account's role. You can then map that UUID to the list of roles to determine the account's role.

Groups

A group is a collection of entities that Turbonomic can work with as a unit. The most common use of groups is to set scope. In the Turbonomic user interface, you can use groups to set scope for:

- Dashboards and Custom Dashboard panels
- Most views in the GUI
- Deployment Profiles (to limit deployment and reservation recommendations)
- Plans
- User accounts
- Workload Placement Segments
- Policy settings
- Scheduled reports

Using the API, you can set scope directly to the following resource items:

- Users
- User groups
- Planer markets

However, with groups you can establish scope for the API activities you want to perform. For example, you can loop through the members of a group to get their historical data. In this way, you can use groups to get the same scoping effects that can be had in the GUI.

Turbonomic uses groups to organize the environment's inventory into categories. It creates two types of groups, which the API identifies by `creationClassName`:

In addition, Turbonomic creates deployItems objects that group together the VMs that are to be set aside for a reservation or a deployment action.

Group Types

The following URL gets a listing of all the groups on the appliance:

`https://guest:guest@MyVmt.com/vmturbo/api/groups`

The returned list includes different types of groups, with each type identified by the item's `creationClassName`. The group types include:

- `Folder`
  These appear in the user interface as folders, and are for visual organization. Discovered folders represent the folder structure in the target — for example, the vCenter folder structure. In addition, Turbonomic discovers vCenter Server resource pools, and groups them into folders.
- `Group`
  Turbonomic places discovered entities into standard groups, and users can create groups of their own with static or dynamic membership.
- `RefGroup`
  A user-defined group of groups — for example, a group of PM cluster groups.
- `Cluster` and `StorageCluster`
  Groups that corresponds to discovered clusters.
- `DiscoveredGroup`
  Groups that are defined by a target service. For example, Turbonomic can discover DRS domains that were defined in vCenter Server.

- `MarketGroup`
  A group that is based on the infrastructure cost of that entity type. These groups are based on the Infrastructure Cost settings in the Turbonomic policies.
- `StaticMetaGroup` and `MetaGroup`
  Groups used internally by Turbonomic. The API does not return any members of these groups.

Understanding the different group types helps you find the group you want. If you know the name of a specific group, then you can filter the returned list for a group of that name. Or if you want to step through all the clusters Turbonomic has discovered, you can filter all the entries with a `creationClassName` of `Cluster` or `StorageCluster`.

Creating Groups

Turbonomic users can create static and dynamic groups. You can also create static and dynamic groups with the API. When you create a group, you pass a boolean flag, where `true` indicates a static group.

For a static group, you specify the entity type and an array of entities as group members. Note that the member entities must all be of the same type. You cannot mix VMs and hosts in the same group, for example.

To create a dynamic group, you specify the type of entity to add to the group, as well as the match criteria to use for that group. Then you specify a regular expression for a match of the given criteria. For example, you can specify a group of VMs, with match criteria of `vmsByPMName`. If you then give `dev.*` as the match expression, the group will contain all VMs that are hosted on a PM with a name that begins with `dev`.

For more information, see groups on page 43.

Managed Reservations

Turbonomic includes the capability to reserve resources for VMs you plan to deploy in the future. This feature includes calculating the placement for these VMs, and then managing reserved copies of them so users can see their placement in the GUI, and plans can include these reserved VMs in their calculations. Users can see that a reservation is in an active or pending state. Users can also direct Turbonomic to deploy the VMs from an active reservation.

Note that the list of reservations is external to the markets that might be current on the appliance. However, when you deploy a reservation, that action deploys the VMs in the physical environment, and so they will get added to the real-time market.

With the API, you can perform the following:

- Get list of reservations
- Get list of reservation members
- Create reservations
- Execute reservation actions (deploy reserved vms)
- Delete a current reservation

Getting Reservation Data

The API can get a list of all the current reservations. You can filter the returned list by reservation state (reserved or pending). For a given reservation you can also get the list of member reserved VMs, identifying the host PM and datastore for each.

Getting a List of Reservations

The following GET call gets a list of all the reservations currently on the appliance. Notice that the URL does not include the UUID of any reservation:

```
https://guest:guest@10.10.172.62/vmturbo/api/reservations
```

In the returned list, each item includes the following information:

- `catalogItem`
  The `name` or `uuid` of the deployment profile used for this reservation.
- `displayName`
  The reservation's display name.
- `expirationDate`
  The date when the reservation expires.
- `name`
  The reservation's internal name.
- `profileName`
  The `name` or `uuid` of the VM template used for this reservation
- `startDate`
  The date when this reservation will deploy.
- `vmCount`
  The number of VMs to reserve.
- `vmPrefix`
  The name prefix assigned for the VMs created by this reservation
- `status`
  The current status of the reservation
  - `DEPLOYING` — Turbonomic is deploying the reserved VMs
  - `DEPLOY_SUCCEEDED` — The reservation was successfully deployed
  - `LOADING` — Turbonomic is calculating the placement of a reservation
  - `IN_PROGRESS` — Deployment in progress
  - `RESERVED` — Active reservation
  - `PENDING` — Pending reservation
  - `PLACEMENT_SUCCEEDED` — For a newly created reservation, the environment has sufficient resources to place the workload; you can execute a `reserve` action to put this in the `RESERVED` state
  - `PLACEMENT_FAILED` — For a newly created reservation, the environment doesn't have resources to place the workload; you can execute a `reserve` action to put this in the `PENDING` state
  - `RETRYING` — Turbonomic is trying to place the workload of a pending reservation
- `uuid`
  The UUID of this reservation.

Getting a List of Reservation Members

If you pass the UUID of a reservation to the resource, the call returns a list of reserved VMs. Each list item shows placement, identifying the datastore and host specified for the reservation. Note that you must have executed the placement action in order for the list to include placement information.

The following GET call gets the member VMs in a reservation:

```
https://guest:guest@10.10.172.62/vmturbo/api/reservations/_QxGdkTdlEeSvatd2DB2axg
```

Creating a Reservation

To create a reservation via the REST API, perform the following steps:

1. **Identify the deployment profile and template you will use.**
   Execute GET calls to the templates and services resources. Use the results to choose which template and deployment profile you will use. Every deployment profile includes a list of templates that it is specified to work with. The easiest approach is to get the deployment profile you want, and then choose a template from the list or related items. For more information, see templates on page 50and deploymentprofiles on page 55.
2. **Execute the POST call to create the reservation.**
   Use the UUIDs or names of the template and deployment profile you have chosen. Provide the other required information, and execute the call.
   On success, this call returns the UUID of the created reservation.
3. **Optionally, review the reservation details.**
   You can get the list of reservations, and then inspect the entry that matches the UUID you got when you created the reservation. This will show you the status of the reservation — The initial status of this reservation should be PLACEMENT_SUCCEEDED. If the placement failed, then there are not enough resources for the VMs you want to reserve — you can `retry` the reservation after conditions have changed.
   You can also get the entry for this specific reservation. That returns a list of reserved VMs. Initially, the list of VMs should not have datastores or hosts assigned to the VMs.
4. **To complete the reservation, execute the `reserve` action.**
   Use the UUID that was returned when you created the reservation, and execute a POST call with `action=reserve`.
   The call returns true on success. You can then inspect the reservation to see how it's changed. The status should now be RESERVED, and the list of member VMs should include datastore and host names to indicate their placement.

The following POST call creates a reservation:

```
https://guest:guest@10.10.172.62/vmturbo/api/
reservations?reservationName=NewRes&vmPrefix=New_VMs_&count=3&templateName=T420f26cf-
bbe6-28d6-e212-ddd859cc8ef3&deploymentProfile=_w9S90DK7EeSvatd2DB2axg&deployDate=10-10-
2014 00:00:00
```

Executing Reservation Actions

For a given reservation, you can execute the following actions:

- reserve
  Convert a proposed reservation into an active reservation.
- deploy
  Deploy the reserved VMs. You can only execute this action on a reservation in the RESERVED state.
- retry
  For a reservation that is in the PENDING state, you can execute this action to try placing the VMs again. Usually, you perform this action after conditions of your environment have changed. For example, if you add a new host to the environment you should retry the reservation.

The following POST call executes the `reserve` action on a reservation:

```
https://guest:guest@10.10.172.62/vmturbo/api/reservations/
_QxGdkTdlEeSvatd2DB2axg?action=reserve
```

Deleting Reservations

Deleting a reservation removes it from the appliance, and frees up any resources that were devoted to placing its member VMs. The following URL deletes the identified reservation:

```
https://guest:guest@10.10.172.62/vmturbo/api/reservations/_TeLG0DdwEeSvatd2DB2axg
```

Entity Templates

Templates describe entities in terms of the resources they will provide or consume. Users can create templates, and Turbonomic also discovers templates that have been created by a target service.

Turbonomic uses templates to describe entities for:

- Plans
  Templates describe workloads (VMs) or providers (hosts or datastores) that you can add to a plan.
- Headroom calculations
  Turbonomic uses VM templates to calculate headroom in the Cluster Capacity dashboard.
- Infrastructure Cost
  You can identify templates to use when dividing the inventory into groups based on relative cost.
- Deployment and Reservations
  You specify a template to characterize the VMs you want to add to the inventory via reservations and deployments.

For deployments, Turbonomic uses templates to describe the VMs, and also uses deployment profiles to identify the physical files that will be copied to deploy a VM, as well as optional placement policies.

This section describes `templates` and `deploymentprofiles` resources.

NOTE: Turbonomic can discover VM template data on hypervisors to use in deploy actions. Discovered templates use the following naming convention for the display name:`<IP Address>::TMP-<Template Name>`where the `IP address` identifies which hypervisor the template was discovered on (for example, `displayName="10.10.172.203::TMP-Suse Template"`). You should never edit or delete discovered templates. Turbonomic also generates discovered deployment profiles that are associated with discovered templates. These deployment profiles include a `discovered=true` attribute. You cannot edit discovered deployment profiles.

Templates

The following URL gets a listing of all the templates on the appliance:

`https://guest:guest@MyVmt.com/vmturbo/api/templates`

The list can include templates of the following type, where the type is identified in the `creationClassName` attribute.

- `VirtualMachineProfile` — for virtual machines
- `PhysicalMachineProfile` — for physical machines (hosts)
- `StorageProfile` — for datastores

Creating Templates

To create a template, you first choose the type of template to create. To specify the template type, declare it in the qualified template name. A qualified name includes the `creationClassName`, plus the display name, separated by two colons. For example, `Storage::Large` is the qualified name for a Storage template named Large.

To define the template, provide values for the parameters your template type requires — Each type of template requires a different set of parameters. You must provide values for all the parameters of the given type, except the `templateUuid` parameter. For the full list of template parameters, see POST: templates.

Editing Templates

To edit a template, provide a UUID for the template in the parameters. You must then provide values for all the other parameters for that template.

Deployment Profiles

The API only supports a GET method for deployment profiles. The following URL gets a listing of user deployment profiles that are on the appliance:

`https://guest:guest@MyVmt.com/vmturbo/api/deploymentprofiles`

Each item in the returned list can contain the following:

The opening tag for the deployment profile:`<TopologyElement creationClassName="ServiceCatalogItem" discovered="false" displayName="DEP-DSL-4.4.10" name="DSL-4.4.10" uuid="_yRFoAP2zEeGFKuqOMSqaNw">`

The location of the physical files used to deploy an instance of the associated template:<uris>http://download3.vmware.com/software/vma/vMA-ovf-4.0.0-161993.ovf</uris>

The deployment scope — users can specify datacenter or cluster scope to limit where the VM can be deployed:<scope>04e9adc245fe11494115af8740cf95c7aad9994e</scope>

The templates that are related to this deployment profile:<TopologyRelationship childrenUuids="T420f54c7-e3a6-2831-3242-c98df991e8a9" name="RelatedProfiles"/>

The closing tag for the deployment profile:</TopologyElement>

To Do Lists

The To Do list specifies the recommended actions for you to perform in your environment, and for each action it includes the associated notification — the reason for the action. In the GUI, a user can toggle the Action List display to show the recommended actions or the notifications (called Risks and Opportunities in the GUI).

Getting Action Logs

The API uses the `actionlogs` resource to expose To Do lists. The following URL gets a list of all the To Do lists current on the appliance:

```
https://guest:guest@MyVmt.com/vmturbo/api/actionlogs
```

Turbonomic maintains separate To Do lists for each market. This includes lists for the real-time market, internal-use planning markets (for example, the Capacity Planning market), and markets for any user-created plans. When you get a list of `actionlogs` items, the `displayName` and `name` attributes indicate the associated market. For example, assume the following list of `actionlogs` items:

```
<TopologyElements>
<TopologyElement creationClassName="ActionLog" displayName="ActionLog/Market_Default"
name="Market_Default_ActionLog" uuid="_TPRJkC-YEeSsp7EWySXKWQ"/>
<TopologyElement creationClassName="ActionLog" displayName="ActionLog/Market_Capacity"
name="Market_Capacity_ActionLog" uuid="_VSv9QjJWEeSsp7EWySXKWQ"/>
<TopologyElement creationClassName="ActionLog" displayName="ActionLog/Market"
name="Market_ActionLog" uuid="_TPjdcS-YEeSsp7EWySXKWQ"/>
</TopologyElements>
```

This listing has To Do lists for three markets:

- `Market_Default`
  The template market. This market is never in a running state, so the To Do list will always be empty.
- `Market_Capacity`
  The market to calculate headroom in the Cluster Capacity dashboard. This is an internal-use plan.
- `Market`
  The real-time market

NOTE: For more information about market types, see groups on page 43.

Getting Action Items

For a given `actionlogs` item, you can get the list of actions. Each action item contains attributes to describe the action, followed by the associated notification element. The action attributes include:

- `acceptTime`
  If the action was accepted, the time that Turbonomic executed the action.
- `recommendTime`
  When the action was posted to the list.
- `actionType`
  For example, MOVE, CHANGE, etc.
- `category`
  One of `Prevention`, `Performance Assurance`, `Compliance`, or `Efficiency Improvement`.
- `current`
  The FROM item for the action.
- `currentClassName`
  The entity type for the FROM item.
- `new`
  The TO item for the action.
- `newClassName`
  The entity type for the TO item.
- `target`
  The entity that will be affected. For example, for a VM move, the target VM is moved from the current host to the new host.
- `actionType`
  For example, MOVE, CHANGE, etc.
- `targetClassName`
  The entity type for the target.
- `targetUuid`
  The target's UUID.
- `explanation`
  The explanation that appears in the To Do list.
- `shortDescription`
  A short version of the explanation.
- `problem`
  The problem description that appears in the To Do list.

- `progress`
  For an action that is currently being executed, a percentage of its completion.
- `savings`
  An indication of how much improvement this action will yield in your environment.
- `uuid`
  The action item's UUID.

Executing Actions

To execute an action, get the action item you want to execute from the real-time market's `ActionLog`, then post a request to that action item with an action parameter set to either `accept` or `reject`.

On success, the API returns `success`.

Logging

Turbonomic maintains two logs that are related to the operation of the installed appliance:

- Notifications
- Audit log

You can access these logs to inspect current or historical Turbonomic status, or to trigger actions in external applications.

Notifications

Turbonomic maintains a list of the problems and notifications that arise in your environment. The types of notification are:

- `MarketProblem`
  Issues Turbonomic identifies within your virtual environment.
- `Discovery`
  Issues that occur as Turbonomic performs discovery.
- `Monitoring`
  Issues that affect Turbonomic as it monitors your environment.
- `Control`
  Issues that affect Turbonomic as it performs recommended actions.
- `Mediation`
  Communication issues that arise when Turbonomic sends commands to discover, monitor, or change your environment.
- `Healthcheck`
  Issues that affect Turbonomic performance. These issues are discovered via periodic Turbonomic health check tests.
- `InterAppliance`
  Issues that occur on an aggregating appliance as Turbonomic communicates with target Turbonomic appliances.

The following URL gets a list of notifications:

```
https://guest:guest@MyVmt.com/vmturbo/api/notifications
```

To refine the query, you can pass the notification type, a flag for active or inactive notifications, and start/end times for a historical range.

For `MarketProblem` notifications, you can get the related action items. Pass the UUID of the given notification, followed by the actionitems resource.

For example, the following URL gets a list of action items associated with the passed notification:

```
https://guest:guest@MyVmt.com/vmturbo/api/notifications/_XHDcAS-vEeSpLNszcJUzcw/
actionitems
```

The result is a list of `ActionItem` elements includes a description of the action, and the notification item.

Audit Logs

Turbonomic maintains a log of events that occur in your environment. The audit log records the initiation, execution and completion of the actions triggered by Turbonomic, and includes entries for each of these events. For example, a VM Move from Turbonomic can include audit log entries for the following:

- Move action recommended (Turbonomic)
- Move action accepted (Turbonomic)
- Move performed (Hypervisor response to Turbonomic command)
- Move completed (Turbonomic)

The audit log would include entries for each of these events.

The following URL gets the audit log:

```
https://guest:guest@MyVmt.com/vmturbo/api/auditentries
```

To get a historical range of audit entries, provide start and end times as follows:

```
https://guest:guest@MyVmt.com/vmturbo/api/
auditentries?starttime=1404777600000&endtime=1409184000000
```

Inventory Summary

The inventory resource gets a summary of the entities in each datacenter in your environment. For each target, the list presents the count of entities in each associated datacenter. You get both the entity type, and the entity state. The datacenters are identified by ID, so you can use the API to get the members of each datacenter. If you call this from an aggregating instance of Turbonomic, you need to determine which underlying server actually manages the target.

Settings Summary

The `settings` resource gets a list of the action modes you have set for different entity types in the groups in your environment. For example, you can use this to quickly get a list of the VMs with resize automated — from this list you can get the group ID, and from that ID you can get the list of member entities. If you call this from an aggregating instance of Turbonomic, you need to determine which underlying server actually manages the target and its associated groups.

Target Management

The Turbonomic GUI provides forms to specify targets, and a chart to display a summary of the entities that have been discovered through these targets. The API exposes these through the `targets` and `discoveryprogress` resources.

Working with Targets

A target is a service that performs management in your virtual environment. Turbonomic uses targets to monitor workload and to execute actions in your environment.

Turbonomic can connect with the following types of targets:

- Hypervisors or other VM management services such as VMWare VCenter, or Citrix XenServer
- Cloud managers such as CloudStack, OpenStack, or vCloud Director
- Load balancers such as Citrix NetScaler
- Storage controllers such as NetApp Storage Systems
- Computing fabrics such as Cisco UCS
- Turbonomic appliances, used as targets for aggregated installations of Turbonomic

---

NOTE: For more information about targets, please be sure to review the Turbonomic online documentation. You should not modify targets without a full understanding f how they work with Turbonomic.

Getting Targets

The API uses the `targets` resource to expose To Do lists. The following URL gets a list of all the targets configured on the appliance:

```
https://guest:guest@MyVmt.com/vmturbo/api/targets
```

The returned items include the target name, display name, target type, and its UUID. For example:

```
<TopologyElements>
<TopologyElement creationClassName="VMTTarget" displayName="10.10.99.211"
name="10.10.99.211" targetType="NetApp" uuid="_n993oA6mEeSWoLQ9fCFMig"/>
...
</TopologyElements>
```

The target types include:

- `vCenter`
- `Hyper-V`
- `XenServer`
- `RHEV`
- `NetScaler`
- `vCloudDirector`
- `CloudStack`
- `VMM`
- `UCS`
- `NetApp`
- `AWS`
- `VMTAppliance`

Executing Discovery

Whenever you edit or add a target, you should execute rediscovery. A POST to the targets resource executes rediscovery for all targets. If you provide a target name or UUID, the POST request rediscovers for that target. Note that the product GUI only provides full rediscovery — with the API you can selectively rediscover individual targets.

Creating and Editing Targets

To create a target, execute a POST request with the parameters that are required for that target type. All targets require:

- `targetType`
- `nameOrAddress`
- `username`
- `password`

Some targets use other parameters — to understand the requirements for each target type, consult the Turbonomic documentation.

To edit an existing target, provide the target's name or UUID in the POST request. If you do not provide a name or UUID, the request creates a new target.

Remember to execute discovery after creating or editing a target.

Deleting a Target

To delete a target, execute a DELETE request to the `targets` resource, giving the name or UUID of the target to delete.

Discovered Entities

The following URL gets a listing of discovered entities have been discovered by the appliance:

`https://guest:guest@MyVmt.com/vmturbo/api/discoveryprogress`

The returned list contains a list similar to the following:

```
<DiscoveryProgressCounts>
<DiscoveryProgressCount className="DataCenter" count="4"/>
<DiscoveryProgressCount className="Storage" count="19"/>
<DiscoveryProgressCount className="Network" count="9"/>
<DiscoveryProgressCount className="VirtualMachine" count="77"/>
<DiscoveryProgressCount className="PhysicalMachine" count="22"/>
</DiscoveryProgressCounts>
```

Scheduled Action Windows

The Turbonomic GUI provides a way to schedule action windows — periods of time that a certain action can be performed for a given scope. The API represents these action windows via the `schedules` resource.

NOTE: An action window does not defer a recommended action to be executed at a later date. The action window is a period of time during which a specific type of action can be automatically executed — *if Turbonomic recommends that action in the To Do list during that action window*. See the User Guide for more information about action windows, and how users define them through the user interface.

An action window includes the following parameters:

- `name`
- `startDate`
  The starting date for the action window's scheduled time range.
- `endDate`
  The ending date for the action window's scheduled time range.
- `startTime`
  The start time for the action, within any day that this action window is to open.
- `endTime`
  The end time for the action, within any day that this action window is to open.
- `color`
  Color is expressed as an integer that represents a HEX color code. The hex prefix is optional. For example, `0066FF` and `0x0066FF` both represent a shade of Blue.
- `group`
  An action window can apply to a specific cluster. This parameter is the ID of the given cluster. If you do not provide a group ID, the action window applies globally.
- `action`
  Depending on the type of entity the action window is for, you can declare a specific type of action. For example, a common action to schedule a window for is `resizeVM`. The documentation for creating and editing action windows includes a full list of actions.
- `actionMode`
  Specify whether the action is manual, or automated.
- `recur`
  A string to specify whether the action window recurs periodically, or is a one-time action window.
- `note`
  A description of the action window.

Provide dates and times in simple format — MM-DD-YYYY and HH:MM. Note that the combination of start date and time must be earlier than the combination of end date and time.

To create a one-time action window (does not recur), specify a value of `NO` for the `recur` parameter. In that case, the schedule will use the start date and start time to open the window.

Creating Scheduled Action Windows

Use the POST method to create a new action window, providing values for the parameters. Most of the parameters are required, however, the following are optional:

- `group`
  If you do not provide a scope for this action window, then it will have a global effect.
- `recur`
  When creating an action window, NO is the default. If you do not provide a value for this parameter, the action will create a one-time action window.

When you create a new action window, the API returns the action window's UID on success.

Editing an Existing Action Window

Use the PUT method to edit an existing action window, providing values for the parameters you want to change. All of the parameters are optional. However, to change a recurring window to a one-time window (does not recur), you must specify a value of NO for the recur parameter. In that case, the schedule will use the start date and start time to open the window.

NOTE: For this release of the API, this is the only use of a PUT method that takes a subset of the parameters. To edit any other resource, you must use POST, and effectively create a new instance of the entity, passing the same values for parameters you don't want to change, and new values for the parameters you do want to change. When you edit an action window, the API uses the parameters you pass, and then creates a new action window — The new action window has a new UID, so you must be sure to use that UID for subsequent calls.

When you edit a new action window, the API returns the action window's UID on success.

REST Resources

The Turbonomic REST API provides a set of resources you can access to request data from the appliance, or to modify data on the appliance (add or delete entities). The REST API supports the following HTTP methods:

- GET: request data about an identified entity
- POST: create new entities, new entity attributes, or change existing entity attributes
- DELETE: remove an entity or attribute

NOTE: The Turbonomic REST API does not use PUT requests to edit existing resources. Instead, you use a POST request to assign new data to an existing entity. POST of an existing entity is destructive, and creates a new instance of that entity. For example, assume you have an existing group that has 10 member entities. If you then send a POST request for the same group with two member entities, the resulting group will have two entities. To incrementally add entities to a group, you would sent a POST request to that group's `entities` resource, adding the ID of the new group member.

Resource Data Returns

Resources return data for GET methods in XML format.

For POST and DELETE methods, and for error conditions, resources return standard HTTP status codes.

Common URL Structure

Every resource URL shares the following characteristics.

Appliance Access

The resources operate on the data that is managed by a given Turbonomic appliance. Appliance access is granted via user name and password. As a result, every resource URL must specify the domain by user name, password, and appliance address, as follows:

```
https://<username:password>@<appliance>/...
```

NOTE: Some browsers do not allow user information such as username and password in an URL. To access REST resources by entering the URL directly in a browser, you can omit this information and provide it when the browser prompts you.

Base URL

Resources are located on the appliance at the following location:

```
https://<username:password>@<appliance>/vmturbo/api
```

Example URLs

The following examples illustrate the common structure of resource URLs.

To get a list of users registered with the appliance:

```
https://<guest:guest>@<123.123.123>/vmturbo/api/users
```

To get a list of all groups that are defined on an appliance:

```
https://<guest:guest>@<123.123.123>/vmturbo/api/groups
```

To get a list of physical machines managed by the appliance on the default market:

```
https://<guest:guest>@<123.123.123>/vmturbo/api/markets/Market/hosts
```

users

The `users` resource manages individual user accounts. You can use it to:

- Get lists of users
- Add new user accounts
- Overwrite an existing account with new data
- Delete user accounts

GET: users

`vmturbo/api/users/[{name_or_uuid}]`

Get a list of users. If you provide a username or uuid, returns the entry for that user.

`{name_or_uuid}`	String
	Optional
	The user's name or unique ID.

EXAMPLE: Get list of users:

URL

`https://guest:guest@123.123.123.12/vmturbo/api/users`

Returns

A listing of all the users defined on the appliance. The following shows an entry for a scoped user:

```
<TopologyElements>
<TopologyElement creationClassName="User" displayName="test2" isScoped="true"
loginProvider="Local" name="test2" userType="DedicatedCustomer"
uuid="_tPTYQJM9EeO3R6_VsS1zYA">
<scope>_we6Fs5JqEeO3R6_VsS1zYA</scope>
<TopologyRelationship childrenUuids="_4UAioQY-Ed-WUKbEYSVIDw" name="role"/>
</TopologyElement>
...
</TopologyElements>
```

POST: users

```
vmturbo/api/users
--data UserName=$username&UserPassword=$userpassword&UserRole=$userrolename
--data UserName=$username&UserPassword=$userpassword&UserRole=$userrolename
&UserType=$usertype&UserScope[]=$group_uuid1&UserScope[]=$group_uuid2
```

Add a new user account to the Turbonomic server. To edit a user account, overwrite an existing user account with a POST request for a user that specifies the same `UserName`, but includes the changes to the other data fields.

You can get a list of supported roles from the `roles` resource (`https://guest:guest@123.123.123.12/vmturbo/api/roles`).

`UserName`
String
Required
The username for this account.

`UserPassword`
String
Required
The password for this account. For an account that uses AD authentication, give LDAP (the account will use its AD password for authentication).

`UserRole`
String
Required
The role for this account. Can be one of:
- observer
- advisor
- automator
- administrator

`loginProvider`
String
Optional
The authentication type for this account. Can be one of:
- Local — The default: Authentication performed on the Turbonomic appliance.
- LDAP — Authentication performed by Active Directory.

`UserType`
String
Required
The type for this account. Can be one of:
- DedicatedCustomer — Default if you do not specify a UserType
- SharedCustomer

`UserScope[]`
Array of IDs
Optional
An array of group UUID values to limit the scope of this user account. If you do not specify a scope, the account will have unlimited access to the environment. A Shared-Customer user must have a specified scope. For example, specify `...UserScope[]=<group_uuid1>&UserScope[]=<group_uuid2>` to assign two groups to this user's scope.

EXAMPLE: Create a user account, setting DedicatedCustomer by default:

URL

```
https://guest:guest@123.123.123.12/vmturbo/api/users
--data UserName=newUser&UserPassword=123456789&UserRole=observer
```

Returns

The entry for the new user, or an empty buffer on failure:

```
<TopologyElements>
<TopologyElement creationClassName="User" displayName="newUser"
name="newCudUser" uuid="_U5b1cMg8EeCypPaggmlo3g"/>
</TopologyElements>
```

DELETE: users

`vmturbo/api/users/{name_or_uuid}`

Delete a user account by the user's name. When creating a user account, this is the value that you pass as `UserName`. When getting user data, this is the value for the `displayName` attribute.

`{name_or_uuid}`	String
	Required
	The user's name or unique ID.

EXAMPLE: Delete the *newUser* user account:

URL

```
https://guest:guest@123.123.123.12/vmturbo/api/users/newUser
```

Returns

The HTTP result.

usergroups

The `usergroups` resource manages Active Directory groups that are defined on the appliance. You can use it to:

- Get lists of user groups
- Add new user groups
- Delete user group

A user group authenticates users via Active Directory groups. Any users that are members of the group can log in using their AD User Principal Names.

NOTE: Group authentication only works if an Active Directory Domain has been specified for the appliance. You should understand how to set AD domains and group accounts in the Turbonomic user interface. For more information, see the Turbonomic online documentation.

GET: usergroups

`vmturbo/api/usergroups/[{name_or_uuid}]`

Get a list of user groups. If you provide a group name or uuid, returns the entry for that group.

`{name_or_uuid}`	String
	Optional
	The group's name or unique ID.

EXAMPLE: Get the list of users that are defined on the Turbonomic server:

URL

`https://guest:guest@123.123.123.12/vmturbo/api/usergroups`

Returns

A list of user groups:

```
<TopologyElements>
<TopologyElement creationClassName="UserGroup" displayName="AdminGroup"
name="AdminGroup" uuid="_4T_7kQY-Ed-WUKbEYSVIDw"/>
<TopologyElement creationClassName="UserGroup" displayName="Guest Group" name="Guest
Group" uuid="_4T_UggY-Ed-WUKbEYSVIDw"/>
...
</TopologyElements>
```

POST: usergroups

```
vmturbo/api/usergroups
--data GroupName=$groupname&GroupRole=$grouprolename
--data GroupName=$groupname&GroupRole=$grouprolename
&GroupScope[]=$group_uuid1&GroupScope[]=$group_uuid2
--data GroupName=$groupname&GroupRole=$grouprolename&GroupType=$grouptype
&GroupScope[]=$group_uuid1&GroupScope[]=$group_uuid2
```

Add a new user group to the Turbonomic server. To edit a user group, overwrite an existing user group with a POST request for a user that specifies the same `GroupName`, but includes the changes to the other data fields.

You can get a list of supported roles from the `roles` resource (`https://guest:guest@123.123.123.12/vmturbo/api/roles`).

`GroupName`

String

Required

The name for this account.

`GroupRole`

String

Required

The role for this account. Can be one of:

- observer
- advisor
- automator
- administrator

`GroupType`

String

Required

The type for this account. Can be one of:

- `DedicatedCustomer`
  Default if you do not specify a `GroupType`
- `SharedCustomer`

`GroupScope`

Array of IDs

Optional

An array of group UUID values to limit the scope of this account. If you do not specify a scope, the account will have unlimited access to the environment. A `SharedCustomer` account must have a specified scope. For example, specify `...UserScope[]=$group_uuid1&UserScope[]=$group_uuid2` to assign two groups to this user's scope.

EXAMPLE: Create a user group:

URL

```
https://guest:guest@123.123.123.12/vmturbo/api/usergroups
--data GroupName=GuestGroup&GroupRole=observer
```

Returns

The entry for the new group, or an empty buffer on failure:

```
<TopologyElements>
<TopologyElement creationClassName="UserGroup" displayName="GuestGroup"
name="GuestGroup" uuid="_U5b1cMg8EeCypPaggmlo3g"/>
</TopologyElements>
```

DELETE: usergroups

Delete a user group by name. When creating a user group, this is the value that you pass as `GroupName`. When getting user data, this is the value for the `displayName` attribute.

`vmturbo/api/usergroups/{name_or_uuid}`

`{name_or_uuid}`	String
	Required
	The group's name or unique ID.

EXAMPLE: Delete the *newGroup* user account:

URL

```
https://guest:guest@123.123.123.12/vmturbo/api/users/newGroup
```

Returns

The HTTP result.

roles

The `roles` resource lists the supported set of roles that can be set to user accounts. You can use it to:

- Get a list of supported user roles
- Get individual user role entries

GET: roles

Get a list of supported roles. If you provide a role name or uuid, returns the entry for that role.

`vmturbo/api/roles/[{name_or_uuid}]`

`{name_or_uuid}`	String
	Optional
	The role name or unique ID.

EXAMPLE: Examples

URL

`https://guest:guest@123.123.123.12/vmturbo/api/roles`

Returns

A list of roles:

```
<TopologyElements>
<TopologyElement creationClassName="UserRole" displayName="Advisor Role" name="advisor"
uuid="_4T_7lwY-Ed-WUKbEYSVIDw"/>
<TopologyElement creationClassName="UserRole" displayName="Observer Role"
name="observer" uuid="_4T_7lQY-Ed-WUKbEYSVIDw"/>
<TopologyElement creationClassName="UserRole" displayName="Automator Role"
name="automator" uuid="_4T_7mQY-Ed-WUKbEYSVIDw"/>
<TopologyElement creationClassName="UserRole" displayName="Administrator Role"
name="administrator" uuid="_4UAioQY-Ed-WUKbEYSVIDw"/>
</TopologyElements>
```

groups

The `groups` resource manages entity groups that are defined on the appliance. You can use it to:

- Get lists of groups
- Get the list of member entities
- Create new static or dynamic groups
- Change the member entities of a static group
- Delete groups

GET: groups — get a list of groups

Get a list of groups. If you provide a `groupname` or `uuid` value, returns the entry for that group.

NOTE: You can call the groups resource as a top-level resource or as a child of a market resource. When you call it from the top level, it lists all the groups on the Turbonomic server. When you call it as a child of a market, it lists the groups within that market.

```
vmturbo/api/groups/[{groupname_or_uuid}]
```

`{groupname_or_uuid}`	String
	Optional
	The internal name of the group, or its unique ID

EXAMPLE: Get the list of groups that are defined on the Turbonomic server:

URL

```
https://guest:guest@123.123.123.12/vmturbo/api/groups
```

Returns

```
<TopologyElements>
<TopologyElement creationClassName="VPool" displayName="WebApp1" name="resgroup-v484"
uuid="016960695832d1f607ce2ba62e3b6c59b1473965"/>
<TopologyElement creationClassName="Group" displayName="VMs_none" name="GROUP-
VMsByNetwork_10.10.172.205\network-235"
uuid="2638244adf3ca112cb06a2d03669e0ad0684a915"/>
...
</TopologyElements>
```

EXAMPLE: Get the entry for a group identified by `name` or `uuid`:

URL

```
https://guest:guest@123.123.123.12/vmturbo/api/groups/resgroup-v484
https://guest:guest@123.123.123.12/vmturbo/api/groups/
016960695832d1f607ce2ba62e3b6c59b1473965
```

Returns:

```
<TopologyElements>
<TopologyElement creationClassName="VPool" displayName="WebApp1" name="resgroup-v484"
uuid="016960695832d1f607ce2ba62e3b6c59b1473965"/>
</TopologyElements>
```

GET: groups — get member entities

Get a list of member entities. For example, get a list of all the physical machines in a group of hosts.

`vmturbo/api/groups/{groupname_or_uuid}/entities`

`{groupname_or_uuid}`	String Required The internal name of the group, or its unique ID

GET: groups — get related items

Get a list of action items or notifications associated with the specified group.

`vmturbo/api/markets/Market/groups/{groupname_or_uuid}/{related_items}`

`{groupname_or_uuid}`	String Required The internal name of the group, or its unique ID
`{related_items}`	String Required The type of related entities to get. Can be one of: - `actionitems` - `notifications`

GET: groups — get workload placement policies

Get a list of the imported placement policies and authored workload placement segments that are on your server. This is a special instance of getting group entities, where `GROUP-VMTSegments` is the name of a standard group in Turbonomic.

```
vmturbo/api/groups/GROUP-VMTSegments/entities
```

EXAMPLE: Get the segments defined on your server:

URL

```
https://admin:admin@123.123.123.12/vmturbo/api/groupsGROUP-VMTSegments/entities
```

Returns

```
<TopologyElements>
<TopologyElement creationClassName="CommSegmentation" displayName="Authored_Segment"
name="Segmentation-GROUP-VMsByNetwork_3ac6f813087adf37f6ae30edb947c461a0e47ef5::GROUP-
PMsByCluster_10.10.150.253\datacenter-21\domain-c26" uuid="_Gmt_0CN0EeahRc5xCzYm_Q" />
</TopologyElements>
```

POST: groups — create a group

Create a new group.

```
vmturbo/api/groups
--data groupName=$groupname
--data groupName=$groupname&seType=$type&reportEnabled=$reportEnabled
&static=$static&uuidsList[]=$uuid1&uuidsList[]=$uuid2
--data groupName=$groupname&seType=$type&reportEnabled=$reportEnabled
&static=$static&xmlId=$match_criteria&expVal=$regexp
```

`groupname`	String Required The display name of the group
`seType`	String Required Identifies the type of service entities that will be members of the group. Can be one of `VirtualMachine`, `PhysicalMachine`, `Storage`, `VirtualDataCenter`, or `Application`.
`reportEnabled`	Boolean Optional One of `true` or `false`. Default is `true`.

`static`

Boolean

Optional

One of `true` or `false`. Default is `false`. Specifies whether to create a static or dynamic list. For a static list, you must provide `seType` and `uuidsList[]` arguments. For a dynamic list you must provide `seType`, `xmlId`, and `expVal`.

`uuidsList[]`

Array of IDs

Optional

ID values for the members of a static group. Cannot be used with `xmlId` and `expVal`, you must provide a value for `seType`, and `static` must be `true`. For example, specify `...uuidsList[]=<item_uuid1>&uuidsList[]=<item_uuid2>` to assign two entities to this static group.

`xmlId`

String

Optional

The type of entities to add as members of a dynamic group. Use this in conjunction with `expVal` to populate members of a dynamic group. Note that `static` must be `false`, and you cannot use this parameter with `uuidsList[]`. The value for `xmlId` can be one of:

- `pmsByCPUModel`
  Add PMs with matching CPU model names
- `pmsByDC`
  Add VMs that are members of datacenters with matching display names
- `pmsByeNumVms`
  Add PMs that host the number of VMs specified in the expression
- `pmsByMem`
  Add PMs with matching memory capacity
- `pmsByModel`
  Add PMs with matching model names
- `pmsByName`
  Add PMs with matching display names
- `pmsByNumCPUs`
  Add PMs with a matching count of CPUs
- `pmsByStorageNetwork`
  Add PMs that connect to storage entities (datastores) with matching display names
- `pmsByTimezone`
  Add VMs running in the given time zone
- `pmsByVendorName`
  Add PMs with matching vendor names
- `storageByName`
  Add Storage entities (datastores) with matching display names
- `diskarrayByName`
  Add disk arrays with matching display names
- `storagecontrollerByName`
  Add storage controllers with matching display names

- switchByName
  Add switches with matching display names
- vmsByDC
  Add VMs that are members of datacenters with matching display names
- vmsByGuestName
  Add VMs that are running a matching guest OS
- vmsByName
  Add VMs with matching display names
- vmsByPMName
  Add VMs hosted by PMs with matching display names
- vmsByStorageNetwork
  Add VMs layered over storage entities (datastores) with matching display names

expVal

String

Optional

Regular expression to populate group members. Use this with xmlId.

For example, to declare a filter criterion that matches VMs by the name of the VM, you express the seType, xmlId, and expVal as follows: seType=VirtualMachine&xmlId=vmsByName&expVal=MyName.*.

EXAMPLE: Create a new static group:

URL

```
https://admin:admin@123.123.123.12/vmturbo/api/groups
--data groupName=MyGroup&seType=VirtualMachine
&reportEnabled=false&static=true&uuidsList[]=423fcadc-eb38-471d-5036-4e385d0b24e1
```

Returns

```
<TopologyElements>
<TopologyElement creationClassName="Group" displayName="MyGroup" name="GROUP-MyGroup"
uuid="__gd7EMivEeCSL--NGzdwiA"/>
</TopologyElements>
```

EXAMPLE: Create a dynamic group with multiple criteria:

Note that the criteria are expressed as pairs of xmlId and expVal, separated by the OR bar ('|'). Also note that the xmlId parameters must match the seType — You cannot include criteria to match entities of a type that is not declared by the seType parameter.

URL

```
https://admin:admin@123.123.123.12/vmturbo/api/groups
--data groupName=MyDynamic&seType=VirtualMachine
&reportEnabled=false&static=false&xmlId=vmsByName&expVal=d.*|vmsByNumCPUs&expVal=4
```

Returns

```
<TopologyElements>
<TopologyElement creationClassName="Group" displayName="MyDynamic" name="GROUP-MyGroup"
uuid="__gd7EMivEeCSL--NGxxwed"/>
</TopologyElements>
```

POST: groups — add member entities to static group

Add member entities to the group. A POST request to add entities is only valid for existing groups that have been created as static groups.

`vmturbo/api/groups/{groupname_or_uuid}/entities/{uuid}`

`{groupname_or_uuid}`	String Required The internal name of the group, or its unique ID
`{uuid}`	ID string Required The unique ID for the entity you want to add as a member of the group. You add group members one at a time, using a separate POST transaction for each.

EXAMPLE: Add an entity to a static group:

URL

```
https://guest:guest@123.123.123.12/vmturbo/api/groups/GROUP-IncrementalAdd/entities/
423f3bd9-4668-1036-f5fe-07be51d4d8cf
```

Returns

```
<TopologyElements>
<TopologyElement creationClassName="VirtualMachine" displayName="Saba-test-1"
name="vm-1186" uuid="423f3bd9-4668-1036-f5fe-07be51d4d8cf"/>
</TopologyElements>
```

DELETE: groups — delete a group

Delete a group by name or unique ID.

`vmturbo/api/groups/{groupname_or_uuid}`

`{groupname_or_uuid}`	String Required The internal name of the group, or its unique ID

EXAMPLE: Delete a group:

URL

```
https://guest:guest@123.123.123.12/vmturbo/api/groups/GROUP-MyGroup
```

Returns

- `success` for success
- `Failed to delete group` for failure

DELETE: groups — delete member entities

Remove member entities from the group. A DELETE request to remove entities is only valid for existing groups that have been created as static groups.

`vmturbo/api/groups/{groupname_or_uuid}/entities/{uuid}`

`{groupname_or_uuid}`	String Required The internal name of the group, or its unique ID
`{uuid}`	ID string Required The unique ID for the entity you want to remove from the group. You remove group members one at a time, using a separate DELETE transaction for each.

templates

The `templates` resource manages templates that have been discovered in your environment, or defined on the Turbonomic server for use with the Turbonomic and deployment actions. You can use this resource to:

- Get lists of templates
- Create new templates
- Delete templates

NOTE: Turbonomic can discover VM template data on hypervisors to use in deploy actions. Discovered templates use the following naming convention for the display name:`<IP Address>::TMP-<Template Name>`where the `IP address` identifies which hypervisor the template was discovered on. For example:`displayName="10.10.172.203::TMP-Suse Template"` You should never delete these templates.

Turbonomic supports the following types of templates — the returned data identifies template type by the `creationClassName` attribute, as follows:

Template Type	creationClassName
VM Template	`VirtualMachineProfile`
Host Template	`PhysicalMachineProfile`
Storage Template	`StorageProfile`

These types of templates include different resources, which are expressed by attributes in the returned data. These attributes match the resources you can see in the Turbonomic user interface for the given type of entity.

GET: templates

`vmturbo/api/templates/[{name_or_uuid}]`

Get a list of templates. If you provide a `qualifiedTemplateName` or `uuid` value, returns the entry for that template.

`{name_or_uuid}`

String

Optional

The qualified name or the unique ID of the template. A qualified name includes the class name for the type of entity the template represents and the display name, separated by two colons. For example, `Storage::Large` is the qualified name for a Storage template named Large. The class names you can provide are `VirtualMachine`, `PhysicalMachine`, and `Storage`.

EXAMPLE: Get the list of templates that are defined on the Turbonomic server:

URL

```
https://guest:guest@123.123.123.12/vmturbo/api/templates
```

Returns

```
<TopologyElements>
<TopologyElement creationClassName="StorageProfile" displayName="Medium"
name="Storage::Medium" uuid="_v0Ri4cpiEd-hypXfJzX8Wg"/>
<TopologyElement creationClassName="PhysicalMachineProfile" displayName="Small"
name="PhysicalMachine::Small" uuid="_v0Q70MpiEd-hypXfJzX8Wg"/>
<TopologyElement creationClassName="PhysicalMachineProfile" displayName="Medium"
name="PhysicalMachine::Medium" uuid="_v0Q70cpiEd-hypXfJzX8Wg"/>
<TopologyElement creationClassName="VirtualMachineProfile"
displayName="10.10.172.203::TMP-Suse Template" name="VirtualMachine::Suse Template"
uuid="423f3747-9ec6-9270-216a-438ee5af980e"/>
</TopologyElements>
```

POST: templates

`vmturbo/api/templates/{templateName}`

For `VirtualMachine` templates:

```
--data templateUuid=$uuid&numVCPUs=$int&vMemSize=$int&vStorageSize=$int
&networkThroughputConsumed=$int&ioThroughputConsumed=$int&accessSpeedConsumed=$int
&memConsumedFactor=$int&cpuConsumedFactor=$int&storageConsumedFactor=$int
&vendor=$string&model=$string&desc=$descriptionString
```

For `PhysicalMachine` templates:

```
--data templateUuid=$uuid&numCores=$int&cpuCoreSpeed=$int&memSize=$int
&networkThroughputSize=$int&ioThroughputSize=$int&vendor=$string
&model=$string&desc=$descriptionString&price=$int
```

For `Storage` templates:

```
--data templateUuid=$uuid&storageSize=$int&accessSpeed=$int
```

Create a new template, or update an existing template.

When specifying data for this POST request:

- The data you provide is different according to the class name (type) of the template you're creating
- To update an existing template, you must provide every parameter for the given template type
- To create a new template, you must provide every parameter for the given template type, *except* the `templateUuid` parameter

`{templateName}` — String

Required

The name of the template to create or update. A qualified template name includes the class name for the type of entity the template represents and the display name, separated by two colons. For example, `Storage::Large` is the qualified name for a Storage template named Large. The class names you can provide are `VirtualMachine`, `PhysicalMachine`, and `Storage`.

`templateUuid`	Unique ID Optional The unique identifier for the template you want to edit. Only provide a UUID value to edit an existing template — when creating a new template you must omit this parameter.
`numVCPUs`	Integer Required — VirtualMachine The number of VCPUs for the templated VM.
`vMemSize`	Integer Required — VirtualMachine The vMEM for the templated VM.
`vStorageSize`	Integer Required — VirtualMachine The vStorage for the templated VM.
`networkThroughputConsumed`	Integer Required — VirtualMachine The network throughput consumption for the templated VM.
`ioThroughputConsumed`	Integer Required — VirtualMachine The IO throughput consumption for the templated VM.
`accessSpeedConsumed`	Integer Required — VirtualMachine The storage access speed consumption for the templated VM.
`memConsumedFactor`	Float Required — VirtualMachine The memory utilization index for the templated VM.
`cpuConsumedFactor`	Float Required — VirtualMachine The CPU utilization index for the templated VM.
`storageConsumedFactor`	Float Required — VirtualMachine The storage utilization index for the templated VM.
`numCores`	Integer Required — PhysicalMachine The number of cores for the templated PM.

`cpuCoreSpeed`	Integer Required — PhysicalMachine The CPU speed for the templated PM in Mhz.
`memSize`	Integer Required — PhysicalMachine The RAM capacity for the templated PM.
`networkThroughputSize`	Integer Required — PhysicalMachine The network throughput for the templated PM.
`ioThroughputSize`	Integer Required — PhysicalMachine The IO throughput for the templated PM.
`storageSize`	Integer Required — Storage The storage capacity for the templated datastore.
`accessSpeed`	Integer Required — Storage The network throughput for the templated datastore.
`vendor`	String Required The vendor for the templated device.
`model`	String Required The model for the templated device.
`price`	String Required The price for the templated device.
`desc`	String Required The description for the templated device.

EXAMPLE: Create a new PhysicalMachine template:

URL

```
https://cud:cud@10.10.172.88/vmturbo/api/templates/PhysicalMachine::MediumPhysMach
--data numCores=7&cpuCoreSpeed=1234&memSize=1234&networkThroughputSize=1234
&ioThroughputSize=1234&vendor=MyCo&model=ARF&desc=My medium template&price=100
```

Returns

```
<TopologyElements>
<TopologyElement creationClassName="PhysicalMachineProfile"
displayName="MediumPhysMach" name="PhysicalMachine::MediumPhysMach"
uuid="_LgcSAIMWEeGP9PVqUQfiHg"/>
</TopologyElements>
```

DELETE: templates

Delete a template by name or unique ID.

`vmturbo/api/templates/{name_or_uuid}`

`{name_or_uuid}` String

Required

The name or the unique ID of the template. A qualified template name includes the class name for the type of entity the template represents and the display name, separated by two colons. For example, `Storage::Large` is the qualified name for a Storage template named Large. The class names you can provide are `VirtualMachine`, `PhysicalMachine`, and `Storage`.

EXAMPLE: Delete a template:

URL

```
https://cud:cud@10.10.172.88/vmturbo/api/templates/Storage::MyMedium
```

Returns

- `true` for success
- For request to delete non-existing template, response does not contain any data
- Failed HTTP request code failure

deploymentprofiles

The `deploymentprofiles` resource manages deployment profiles that have been discovered in your environment, or defined on the Turbonomic server for use with deployment actions.

You can use this resource to:

- Get lists of deployment profiles

NOTE: Turbonomic can discover VM template data on hypervisors to use in deploy actions. This data includes information that also maps to a Turbonomic deployment profile. In that case, Turbonomic creates a discovered deployment profile to accompany the discovered VM template. Deployment profile data includes the `discovered` attribute, which is `true` for a discovered deployment profile. Discovered profiles also use the following naming convention for the display name:`DEP-<Profile Name>`For example, `displayName="DEP-Win-7_64bit"` You should never delete these deployment profiles.

GET: deploymentprofiles

```
vmturbo/api/deploymentprofiles/[{name_or_uuid}]
DEPRECATED: vmturbo/api/services/[{name_or_uuid}]
```

Get a list of deployment profiles. If you provide a `name` or `uuid` value, returns the entry for that profile.

`{name_or_uuid}`	String
	Optional
	The `name` attribute of the deployment profile, or its unique ID

EXAMPLE: Get the list of deployment profiles that are defined on the Turbonomic server:

URL

```
https://guest:guest@123.123.123.12/vmturbo/api/deploymentprofiles
```

Returns

```
<TopologyElements>
<TopologyElement creationClassName="ServiceCatalogItem" displayName="DEP-RHEL_64bit"
name="RHEL_64bit" uuid="_fWW-MshoEeKtxaktNqwCpw">
<TopologyRelationship childrenUuids="T4206c729-ec62-6950-d363-d139b2be3a22"
name="RelatedProfiles"/>
</TopologyElement>
<TopologyElement creationClassName="ServiceCatalogItem" displayName="hvDProfile"
name="" uuid="_W8PoAPxZEeGks_RejxacVA">
<TopologyRelationship
childrenUuids="_uRb8IL8BEeK0vZOEcwQ|_8GuXEPxYEeGks|_UKsnkname="RelatedProfiles"/>
</TopologyElement>
</TopologyElements>
```

POST: deploymentprofiles

```
vmturbo/api/deploymentprofiles/[{profileUuid}]
--data name=$string&scope=$uuid&uri=$uriString
&relatedTemplates[]=$uuid1&relatedTemplates[]=$uuid2
```

Create a new deployment profile, or update an existing deployment profile.

`{profileUuid}`	Unique ID Optional The unique identifier for the deployment profile you want to edit. Only provide a UUID value to edit an existing deployment profile — when creating a new deployment profile you must omit this parameter.
`name`	String Required The name of the deployment profile to create or update.
`scope`	Uique ID Optional — Cluster group or datacenter The UUID for a cluster or a datacenter that constrains where associated VMs can be deployed.
`uri`	Uri string Optional A path to the OVF or img.vhd file for the VM that this profile will deploy.
`relatedTemplates[]`	Array of IDs Required — One or more templates ID values for the templates that are associated with this deployment profile.

EXAMPLE: Create a new deployment profile:

URL

```
https://cud:cud@10.10.172.88/vmturbo/api/deploymentprofiles
--data name=MyDeploymentProfile&scope=54a523d51367cac0ea0ba263952249e664afac29
&uri=C:\\ClustStorage\\Volume1\\img.vhd
&relatedTemplates[]=_29UisJj2EeCC5LboJXwW3g
&relatedTemplates[]=_wIBz8Jj4EeC6nYMiQT1jqA
```

Returns

On success, returns `true`.

DELETE: deploymentprofiles

`vmturbo/api/deploymentprofiles/{uuid}`

Delete the deployment profile that is identified by the UUID.

NOTE: Before deleting a deployment profile, you should make sure that it is not currently in use for an existing reservation, and you should make sure that it is not a discovered deployment profile. To check whether the deployment profile is in use, get the list of existing reservations and verify that they don't use the deployment profile — the `profileName` attribute contains the UUID of each reservation's deployment profile. To make sure the deployment profile is not discovered, inspect it via the API — the deployment profile data should include the attribute, `discovered="false"`.

`{uuid}`	String
	Required
	The unique ID attribute of the deployment profile

EXAMPLE: Delete the identified deployment profile:

URL

```
https://guest:guest@123.123.123.12/vmturbo/api/deploymentprofiles/
_W8PoAPxZEeGks_RejxacVA
```

Returns

`true` on success or `false` on failure.

markets

The `markets` resource provides access to all the entities that are managed by the appliance. In the appliance, a market is a management domain. A single appliance can have more than one market. You can use the markets resource to:

- Get lists or instances of markets
- Get information about a market:
  - Projections (from the Project Cluster Resources dashboard)
  - Capacity (from the Cluster Capacity dashboard)
  - Comparison (from the Optimize view)
- Modify an existing planner market
- Deploy VMs (related to the Deploy view)
- Delete planner markets

NOTE: A market describes a virtual environment, including all the entities in that environment's inventory. This section describes API calls that pertain to the entire market. For calls to access entities within a market, see Market Entities on page 67.

At any time, your Turbonomic appliance can have a number of markets in memory. These include a real-time market, and can include a number of planner markets. You can safely perform GET requests on either type of market. However, a POST or DELETE request on the real-time market will modify the real-time data Turbonomic uses to manage your environment. Before executing POST or DELETE requests on markets, you should understand the different market types, and issues associated with operating on them. Please read Accessing Markets on page 8before continuing.

NOTE: To make POST or DELETE requests on a market, the URL login credentials must be for an account with an `advisor`, `automator`, or `administrator` role.

GET: markets — get markets

`vmturbo/api/markets/[{name_or_uuid}]`

Get a list of markets. If you provide a `marketname` or `uuid`, returns the entity for that market.

`{name_or_uuid}`	String
	Optional
	The name of the market, or the market's unique ID.

EXAMPLE: Get the list of markets:

URL

`https://user:pwd@10.10.172.88/vmturbo/api/markets`

Returns

```
<TopologyElements>
<TopologyElement creationClassName="Market"
displayName="_VWg_wHgUEeGGYt0yqL27hQ_BasePlan" name="_VWg_wHgUEeGGYt0yqL27hQ_BasePlan"
state="STOPPED" uuid="_ie5w0IMhEeG2TOYe0j0wkA"/>
<TopologyElement creationClassName="Market" displayName="Market_Default"
name="Market_Default" state="STOPPED" uuid="_GFRthYh9Ed-Sl7QXyf-Qdv"/>
<TopologyElement creationClassName="Market" displayName="Market" name="Market"
state="RUNNING" uuid="_GFRGcYh9Ed-Sl7QXyf-Qdw"/>
</TopologyElements>
```

GET: markets — get cluster projections

`vmturbo/api/markets/{name_or_uuid}/[{entity_type}]/[{cluster_uuid}]/projections`

Gets data that matches the Project Cluster Resources dashboard in the appliance GUI.

`{name_or_uuid}`	String Required The name of the market, or the market's unique ID.
`{entity_type}`	String Optional If you choose to limit the comparison to a single cluster, the type of entity to query. Can be one of: - `groups` - `virtualmachines` - `hosts` - `datastores`
`{cluster_uuid}`	String Optional If you choose to limit the comparison to a single cluster, the cluster's UUID.

EXAMPLE: Get cluster projection data:

URL

```
https://user:pwd@10.10.172.88/vmturbo/api/markets/Market/projections
```

Returns

```
<MarketSummaryItems>
<MarketSummaryItem clusterName="PMs_QualityAssurance\Cluster-1"
clusterUUID="64f2db2b49db3364f1e93fa1d9e360982503a5ba" completeData="false"
currentPMs="2" currentSTs="3" currentVMs="1" lastRun="1374026400689" projectionDays="0"
time="Today">
<MarketSummaryItem clusterUUID="64f2db2b49db3364f1e93fa1d9e360982503a5ba"
currentPMs="2" currentSTs="1" currentVMs="2" projectionDays="30" time="8/2013"/>
<MarketSummaryItem clusterUUID="64f2db2b49db3364f1e93fa1d9e360982503a5ba"
currentPMs="2" currentSTs="1" currentVMs="2" projectionDays="60" time="9/2013"/>
<MarketSummaryItem clusterUUID="64f2db2b49db3364f1e93fa1d9e360982503a5ba"
currentPMs="NA" currentSTs="NA" currentVMs="NA" projectionDays="90" time="10/2013"/>
<MarketSummaryItem clusterUUID="64f2db2b49db3364f1e93fa1d9e360982503a5ba"
currentPMs="NA" currentSTs="NA" currentVMs="NA" projectionDays="180" time="1/2014"/>
<MarketSummaryItem clusterUUID="64f2db2b49db3364f1e93fa1d9e360982503a5ba"
currentPMs="NA" currentSTs="NA" currentVMs="NA" projectionDays="270" time="4/2014"/>

<MarketSummaryItem clusterUUID="64f2db2b49db3364f1e93fa1d9e360982503a5ba"
currentPMs="NA" currentSTs="NA" currentVMs="NA" projectionDays="360" time="7/2014"/>
</MarketSummaryItem>
...
</MarketSummaryItems>
```

GET: markets — get cluster capacity

```
vmturbo/api/markets/{name_or_uuid}/capacity
```

Gets data that matches the Cluster Capacity dashboard in the appliance GUI.

`{name_or_uuid}`	String Required The name of the market, or the market's unique ID.

EXAMPLE: Get cluster capacity data:

URL

```
https://user:pwd@10.10.172.88/vmturbo/api/markets/Market/capacity
```

Returns

```
<ClusterSummaryItemsGroups>
<GroupSummaryItem clusterName="PMs_QualityAssurance\Cluster-1"
clusterUUID="64f2db2b49db3364f1e93fa1d9e360982503a5ba" currentPMs="2" currentSTs="3"
currentVMs="1" date="07/18/2013" headroomVMs="28"
templateName="VirtualMachine::Microsoft_SQL2008-medium"/>
<GroupSummaryItem clusterName="PMs_QualityAssurance\Cluster-2"
clusterUUID="14d94ba08ef417080eb5c2dc7a003b7fa4cdbdab" currentPMs="1" currentSTs="0"
currentVMs="0" date="07/18/2013" headroomVMs="0"
templateName="VirtualMachine::Microsoft_SQL2008-medium"/>
...
</ClusterSummaryItemsGroups>
```

GET: markets — compare current market to desired state

Gets data that compares the current state of market entities to the state they would have if the market accepted all the current recommended actions. Gets data for all entities of the given type.

```
vmturbo/api/markets/{name_or_uuid}/{comparison_type}
```

`{name_or_uuid}`	String Required The name of the market, or the market's unique ID.
`{comparison_type}`	String Required A resource to get the comparison per entity type. Can be one of: - `hostcomparison` - `storagecomparison`

EXAMPLE: Get host comparison data:

URL

```
https://user:pwd@10.10.172.88/vmturbo/api/markets/Market/hostcomparison
```

Returns

```
<ServiceEntities>
<ServiceEntity CPU_utilization_Current="11.99" CPU_utilization_Target="55.97"
IOThroughput_utilization_Current="0" IOThroughput_utilization_Target="0"
Mem_utilization_Current="65.01" Mem_utilization_Target="73.7"
NetThroughput_utilization_Current="0.47" NetThroughput_utilization_Target="0.47"
VMsOnHost_Current="4" VMsOnHost_Target="6" creationClassName="PhysicalMachine"
displayName="hp-esx27.corp.turbonomic.com" marketName="Market" name="host-2614"
priceIndex_Current="8.17" priceIndex_Target="14.46" state_Current="ACTIVE"
state_Target="ACTIVE" uuid="Virtual_ESX_4238b4e8-3652-27f8-441a-19570b077758"/>
<ServiceEntity CPU_utilization_Current="26.97" CPU_utilization_Target="56.22"
IOThroughput_utilization_Current="0" IOThroughput_utilization_Target="0"
Mem_utilization_Current="71.91" Mem_utilization_Target="72.03"
NetThroughput_utilization_Current="0.47" NetThroughput_utilization_Target="0.47"
VMsOnHost_Current="4" VMsOnHost_Target="5" creationClassName="PhysicalMachine"
displayName="hp-esx28.corp.turbonomic.com" marketName="Market" name="host-3077"
priceIndex_Current="12.67" priceIndex_Target="12.78" state_Current="ACTIVE"
state_Target="ACTIVE" uuid="Virtual_ESX_4238a307-8952-ccba-8a64-d5c6f739b2fc"/>
...
</ServiceEntities>
```

GET: markets — compare current market to desired state, per group

Gets data that compares the current state of market entities to the state they would have if the market accepted all the current recommended actions. Gets data for a specified group.

```
vmturbo/api/markets/{name_or_uuid}/group/{group_uuid}/{comparison_type}
```

`{name_or_uuid}`	String Required The name of the market, or the market's unique ID.
`{group_uuid}`	String Required The group's UUID.
`{comparison_type}`	String Required A resource to get the comparison per entity type. Can be one of: - `hostcomparison` - `storagecomparison`

GET: markets — show Optimal Operating Zone data

Gets projection data from the Optimal Operating Zone chart in the Assure Service Performance dashboard.

`vmturbo/api/markets/{name_or_uuid}/target`

`{name_or_uuid}`	String Required The name of the market.

EXAMPLE: Get Optimal Operating Zone data:

URL

`https://user:pwd@10.10.172.88/vmturbo/api/markets/Market/target`

Returns

```
<ServiceEntityGroups>
<ServiceEntityGroupHistory count_CPU_utilization="22.0" count_Mem_utilization="22.0"
count_priceIndex="22.0" current_CPU_utilization="11.18" current_Mem_utilization="20.91"
current_max_CPU_utilization="97" current_max_Mem_utilization="89"
current_max_priceIndex="83.29" current_min_CPU_utilization="0"
current_min_Mem_utilization="4" current_min_priceIndex="1.1" current_priceIndex="5.38"
name="TargetMarket" time="1373994000000"/>
<ServiceEntityGroupHistory count_CPU_utilization="22.0" count_Mem_utilization="22.0"
count_priceIndex="22.0" current_CPU_utilization="11.05" current_Mem_utilization="20.77"
current_max_CPU_utilization="95" current_max_Mem_utilization="88"
current_max_priceIndex="68.36" current_min_CPU_utilization="0"
current_min_Mem_utilization="4" current_min_priceIndex="1.1" current_priceIndex="4.69"
name="TargetMarket" time="1373997600000"/>
<ServiceEntityGroupHistory count_CPU_utilization="22" count_Mem_utilization="22"
count_priceIndex="22" name="TargetMarket" projected_CPU_utilization="9.86"
projected_Mem_utilization="22.23" projected_max_CPU_utilization="56.02"
projected_max_Mem_utilization="73.58" projected_max_priceIndex="11.49"
projected_min_CPU_utilization="1.3" projected_min_Mem_utilization="4.98"
projected_min_priceIndex="1.11" projected_priceIndex="2.95" time="1374163200000"/>
```

POST: markets — creating or modifying a plan

`vmturbo/api/markets/{name_or_uuid}`

To create a plan:

```
--data state=[run|stop]&constraint=[enabled|disabled]
&scope[]=$groupName_or_uuid&scope[]=$groupName_or_uuid...
```

To add clones of templates to a plan:

```
--data state=[run|stop]&constraint=[enabled|disabled]&templateName=$templateName
&count=$int&scope[]=$groupName_or_uuid&scope[]=$groupName_or_uuid...
```

To create or modify a plan, sent a POST request specifying the plan by name. If the plan does not exist, the API will create a new plan. If the plan exists, the API uses the data you pass to modify the plan.

NOTE: You must be careful not to modify the markets named `Market` or `Market_Default`. You can provide any name for a plan in the POST request. If you provide `Market` or `Market_Default`, your Turbonomic session can display unintended consequences. If no plan of the given name exists on the appliance, the API will create a plan of that name. However, if you do not provide a valid name that indicates a user and a plan type, the market will not perform valid functions within the Turbonomic processing.

You can make a POST request to a named planer market to:

- Create a plan (if the named market doesn't exist)
- Run/stop a plan
- Enable/disable a plan
- Set plan scope
- Add clones of templates to the plan

You can also add clones of specific entities to a plan. For information, see POST: entities — add entity copies to plan on page 88.

Note that when modifying an existing plan, you cannot add or remove individual groups. The set of groups you post for scope become the full scope of the plan. If you have set scope to a plan and want to return it to full scope, delete the plan and create a new one without any scope parameters.

`{name_or_uuid}`	String Required The internal name or uuid of the market you will modify.
`state`	String Required Whether to run or stop the plan. Can be one of `run` or `stop`.
`constraint`	String Required Whether to enable processing on this market. Can be one of `enabled` or `disabled`.
`templateName`	String Optional The name of a template that you want to clone when adding workload to this plan.

`count`	String Optional How many instances of the template clones to add.
`scope`	String Optional Array of groups. For each group you specify, the internal name or unique ID of a group to add to the plan's scope. You can only set scope when creating a new plan. For example, specify `...Scope[]=<group_uuid1>&Scope[]=<group_uuid2>` to assign two groups to this plan.

EXAMPLE: Set the scope in a plan:

URL

```
https://usr:pwd@10.10.172.88/vmturbo/api/markets/
_VWg_wHgUEeGGYt0yqL27hQ_BasePlan?scope[]=_uafFIcasEeC9wIEGPhub1G
```

Returns

```
<TopologyElements>
<TopologyElement creationClassName="Market"
displayName="_VWg_wHgUEeGGYt0yqL27hQ_BasePlan" name="_VWg_wHgUEeGGYt0yqL27hQ_BasePlan"
state="READY_TO_START" uuid="_YAfpkITUEeGRw_gjztEB3Q"/>
</TopologyElements>
```

DELETE: markets — delete a planner market

`vmturbo/api/markets/{name_or_uuid}`

Delete the named planner market. For example, if you have set a scope to a plan, and want to remove the scope (set it to a full scope), you must delete the scoped market and create a new one.

`{name_or_uuid}`	String Required The name or unique identifier of the market.

EXAMPLE: Delete a planner market:

URL

```
https://usr:pwd@10.10.172.86/vmturbo/api/markets/_Y7bfIIi2EeGrq73x6zqyeQ_BasePlan
```

Returns

- `200` for success

DELETE: markets — delete entity instance from a plan or replace entity with template

```
vmturbo/api/markets/{name_or_uuid}/entities/{entityname_or_uuid}
--data state=[run|stop]&constraint=[enabled|disabled]&templateName=$templateName
&count=$int
```

For an existing plan, delete the named entity. If you provide template parameters, you can delete the entity and replace it with one or more copies of the specified template.

`{name_or_uuid}`	String Required The internal name or uuid of the market you will modify.
`{entityname_or_uuid}`	String Required The internal name or uuid of the entity to delete from the plan.
`state`	String Required Whether to run or stop the plan. Can be one of `run` or `stop`.
`constraint`	String Required Whether to enable processing on this market. Can be one of `enabled` or `disabled`.
`templateName`	String Optional The name of a template that you want to clone when adding workload to this plan.
`count`	Integer Optional How many instances of the template clones to add.

Market Entities

Each `markets` resource provides access to specific entities. For a given market, you can then access the following types of entities:

- Virtual applications
- Applications
- Containers
- VMs
- Hosts
- Datastores
- Disk arrays
- Storage controllers
- Switches
- Network Groups (VPods and DPods)
- Virtual datacenters
- Datacenters

For each entity type, you can:

- Get lists of entities
- Get instances by name
- Get historical data for an instance
- Get commodity information for an existing instance
    - List of commodities
    - Commodity details
    - Commodity Bought details
- Get actions action or notification information related to an existing instance
    - Related actions
    - Related risks/opportunities (notifications)
    - Related audit log entries
- Clone entities into a plan market from templates or existing instances
- Replace an existing instance in a plan market with a template clone
- Delete existing instances from a plan market

At any time, your Turbonomic appliance can have a number of markets in memory. These include a real-time market, and can include a number of planner markets. You can safely perform GET requests on either type of market. However, a POST or DELETE request on the real-time market will modify the real-time data Turbonomic uses to manage your environment. Before executing POST or DELETE requests on markets, you should understand the different market types, and issues associated with operating on them. Please read Accessing Markets on page 8and Accessing Market Entities on page 11before continuing.

NOTE: To make POST or DELETE requests on a market, the URL login credentials must be for an account with an `advisor`, `automator`, or `administrator` role.

GET: entities — get a list of entities

```
vmturbo/api/markets/{name_or_uuid}/entities
[?classname=$classname]
[&entity=$entity]
[&property=$property]
[&service=$service]
[&resource=$resource]
```

Get a list of entities managed by the market. The real-time market (`Market`) returns all the managed entities on the appliance. A planner market returns all the managed entities that are within the scope of the plan. You can pass parameters to filter the returned results. If the appliance has no instances of the requested entity type, then the resource returns an empty `<ServiceEntities/>` element.

`{name_or_uuid}`

String

Required

The name of the market or the market's unique identifier.

`classname`

String — Regular Expression

Optional

The API returns only entities of the matching class. Valid class names are:

- `VirtualApplication`
- `Application`
- `Container`
- `VirtualMachine`
- `PhysicalMachine`
- `Storage`
- `DiskArray`
- `StorageController`
- `Switch`
- `VPod`
- `DPod`
- `VirtualDatacenter`
- `DataCenter`

`entity`

String — Regular Expression

Optional

The internal name of the entity. The returned data only includes entries with internal names that match this regular expression.

`property`

String

Optional

The property to query. Valid properties are:

- `priceIndex`
- `Produces`
- `ProducesSize`

`service`	String Optional The name of a service provided by the entity. For a list of services this entity provides, get the entity's services resource (`markets/Market/entities/{entityname_or_uuid}/services`).
`resource`	String Optional The name of a resource on the entity. For example, `CPU_utilization`. The returned data excludes all other resources but the named resource. The value string can OR resources together — for example `resource=Mem_utilization|CPU_utilization|IOThroughput_utilization`.

EXAMPLE: Get the list of entities on a named market:

URL

```
https://user:pwd@10.10.172.88/vmturbo/api/markets/_VWg_wHgUEe_BasePlanHWReplace/
entities
```

Returns

```
<ServiceEntities>
<ServiceEntity creationClassName="Storage" displayName="hp-xen6.corp.turbonomic.com/
582c8239-d993-d69c-8be4-83b1129ad4af"
name="com.xensource.xenapi.SR@6000a3f1__VWg_wHgUEe_BasePlanHWReplace" priceIndex="1"
uuid="_GuL2YYM2EeG2TOYe0j0wkA"/>
<ServiceEntity creationClassName="PhysicalMachine" displayName="hp-
xen7.corp.turbonomic.com"
name="com.xensource.xenapi.Host@b3f55ab6__VWg_wHgUEe_BasePlanHWReplace" priceIndex="1"
uuid="_GvXiLIM2EeG2TOYe0j0wkA"/>
<ServiceEntity creationClassName="DiskArray" displayName="DiskArray-hypervdisk"
name="DiskArray-hypervdisk_V" priceIndex="81" uuid="_GuIzLIM2EeG2TOYe0j0wkA"/>
...
</ServiceEntities>
```

GET: entities — by type

```
vmturbo/api/markets/{name_or_uuid}/{type}
[?entity=$entity]
[&property=$property]
[&service=$service]
[&resource=$resource]
```

Get a list of entities of the given type. A planner market returns managed entities that are within the scope of the plan.

`{name_or_uuid}`

String

Required

The name or unique ID of the market.

`{type}`

String

Required

The type of entity to get. Can be one of:

- `entities`
- `virtualapplications`
- `applications`
- `containers`
- `virtualmachines`
- `hosts`
- `hostcomparison`
  Returns the Top-N physical machines, sorted by priceIndex
- `datastores`
- `diskarrays`
- `storagecomparison`
  Returns the Top-N datastores, sorted by priceIndex
- `storagecontrollers`
- `iomodules`
- `switches`
- `vpods`
- `dpods`
- `virtualdatacenters`
- `datacenters`

`entity`

String — Regular Expression

Optional

The internal name of the entity. The returned data only includes entries with internal names that match this regular expression.

`property`

String

Optional

The property to query. Valid properties are:

- `priceIndex`
- `Produces`
- `ProducesSize`

`service`

String

Optional

The name of a service provided by the entity.

`resource`	String Optional The name of a resource on the entity. For example, `CPU_utilization`. The returned data excludes all other resources but the named resource. The value string can OR resources together — for example `resource=Mem_utilization|CPU_utilization|IOThroughput_utilization`.

EXAMPLE: Get a list of host entities with internal names that begin with "com":

URL

```
https://usr:pwd@10.10.172.89/vmturbo/api/markets/Market/hosts?entity=com.*
```

Returns

```
<ServiceEntities>
<ServiceEntity CPU_utilization="11.21" ClusterCommodity_utilization="0"
Cooling_utilization="1" DataCenterCommodity_utilization="0"
IOThroughput_utilization="0" Mem_utilization="17.61" NetThroughput_utilization="0.2"
Power_utilization="1" Produces="4" ProducesSize="4" Q1VCPU_utilization="0"
Q2VCPU_utilization="0" Space_utilization="1" creationClassName="PhysicalMachine"
displayName="hp-xen5.corp.turbonomic.com" name="com.xensource.xenapi.Host@cd13b30"
priceIndex="1.47" uuid="7e2ef488-8e90-49a9-9589-faca88cc8df4"/>
...
</ServiceEntities>
```

GET: entities — entity of type by name or ID

```
vmturbo/api/markets/{name_or_uuid}/{type}/{entityname_or_uuid}
[?property=$property]
[&resource=$resource]
[&service=$service]
[&starttime=$utc_time]
[&endtime=$utc_time]
```

Get data for a named entity of a specified type.

NOTE: You can get groups by name via a given market resource. In that case, the API returns all the group members that are managed by that named market. This can be a different list of members than you get by requesting the groups resource directly from the appliance (`.../groups`).

`{name_or_uuid}`	String Required The name or unique ID of the market.
`{type}`	String Required The type of entity to get. Can be one of: - `entities` - `virtualapplications` - `applications` - `containers` - `virtualmachines` - `hosts` - `datastores` - `diskarrays` - `storagecontrollers` - `iomodules` - `switches` - `vpods` - `dpods` - `virtualdatacenters` - `datacenters`
`{entityname_or_uuid}`	String Required Internal name or ID of the entity. The API returns entries for that entity.
`property`	String Optional One of the following entity properties: `priceIndex`, `Produces`, or `ProducesSize`. If you provide a value, the returned results exclude whichever of these properties you don't specify. The value string can OR properties together — for example `property=Produces|ProducesSize`. If you do not provide a value, this resource assumes `priceIndex|Produces|ProducesSize`. - `priceIndex` - `Produces` - `ProducesSize` - `rOI`
`resource`	String Optional The name of a resource on the entity. For example, `CPU_utilization`. The returned data excludes all other resources but the named resource. The value string can OR resources together — for example `resource=Mem_utilization|CPU_utilization|IOThroughput_utilization`.

`service`	String Optional The name of an entity service. For example, `capacity` of a given resource. The returned data excludes all other services but the named one. You can user regular expressions, and the value string can OR service names together — for example `resource=.*capacity|.*_utilization`.
`starttime`	UTC Time Optional UTC time in ms. The start and end times for a historic range of entries. If you provide no value or an invalid value for `starttime`, this returns the current data. If you provide a value for `starttime`, but not `endtime`, this uses the range from `starttime` to current. If you provide zero for `starttime`, this returns the earliest stored data.
`endtime`	UTC Time Optional UTC time in ms.

EXAMPLE: Get VCPU utilization for a specific GuestLoad application:

URL

```
https://usr:pwd@10.10.172.89/vmturbo/api/markets/Market/applications/GuestLoad-
423fa768-aa0d-11ca-e8a9-d03aa9300547?resource=VCPU_utilization
```

Returns

```
<ServiceEntities>
<ServiceEntity Produces="0" ProducesSize="0" VCPU_utilization="0"
creationClassName="Application" displayName="GuestLoad [Win2003-Web_Clone-10]"
name="GuestLoad-423fa768-aa0d-11ca-e8a9-d03aa9300547" priceIndex="1"
uuid="2c858a4809582ae429b1c01f870e0fb1ba079b58"/>
...
</ServiceEntities>
```

EXAMPLE: Get historical data for a given VM, starting at 00:00 Aug 3, 2012:

URL

```
https://usr:pwd@10.10.172.89/vmturbo/api/markets/Market/virtualmachines/423f94aa-487b-
ca00-181f-d765db224705?starttime=1343952000
```

Returns

```
<ServiceEntities>
<ServiceEntityHistory VCPU_Utilization="23.9" VCPU_peak="159.23"
VCPU_peakUtilization="31.1" VCPU_units="MHz" VCPU_used="122.37" VMem_Utilization="1.2"
VMem_peak="163577.86" VMem_peakUtilization="3.9" VMem_units="KB" VMem_used="50331.65"
VStorage_Utilization="100" VStorage_peak="8181" VStorage_peakUtilization="100"
VStorage_units="MB" VStorage_used="8181" creationClassName="VirtualMachine"
displayName="Win2003-3" name="vm-488" time="1345939200000" uuid="423f94aa-487b-ca00-
```

```
181f-d765db224705"/>
<ServiceEntityHistory VCPU_Utilization="23.2" VCPU_peak="165.89"
VCPU_peakUtilization="32.4" VCPU_units="MHz" VCPU_used="118.78" VMem_Utilization="1.2"
VMem_peak="230686.72" VMem_peakUtilization="5.5" VMem_units="KB" VMem_used="50331.65"
VStorage_Utilization="100" VStorage_peak="8181" VStorage_peakUtilization="100"
VStorage_units="MB" VStorage_used="8181" creationClassName="VirtualMachine"
displayName="Win2003-3" name="vm-488" time="1346025600000" uuid="423f94aa-487b-ca00-
181f-d765db224705"/>
...
</ServiceEntities>
```

GET: entities — provided commodities, by entity name or ID

```
vmturbo/api/markets/{name_or_uuid}/{type}/{entityname_or_uuid}/services
[?property=$property]
[&service=$service]
[?starttime=$starttime]
[&endtime=$endtime]
```

Get commodity details for a named entity of a specified type.

`{name_or_uuid}`
String
Required
The name or unique ID of the market.

`{type}`
String
Optional
The type of entity to get. Can be one of:

- `entities`
- `virtualapplications`
- `applications`
- `containers`
- `virtualmachines`
- `hosts`
- `datastores`
- `diskarrays`
- `storagecontrollers`
- `iomodules`
- `switches`
- `vpods`
- `dpods`
- `virtualdatacenters`
- `datacenters`
- `groups`

`{entityname_or_uuid}`	String Required Internal name or ID of the entity. The API returns entries for that entity.
`property`	String Optional One of the following entity properties: - `priceIndex` - `Produces` - `ProducesSize` - `rOI`
`service`	String Optional The name of a service provided by the entity.
`starttime`	UTC Time Optional UTC time in ms. The start and end times for a historic range of entries. If you provide no value or an invalid value for `starttime`, this returns the current data. If you provide a value for `starttime`, but not `endtime`, this uses the range from `starttime` to current. If you provide zero for `starttime`, this returns the earliest stored data.
`endtime`	UTC Time Optional UTC time in ms.

EXAMPLE: Get the commodities provided by a given host:

URL

```
https://usr:pwd@10.10.172.89/vmturbo/api/markets/Market/hosts/host-340/services
```

Returns

```
<ServiceEntity>
<Commodity displayName="ClusterCommodity/hp-esx42.corp.turbonomic.com"/>
<Commodity displayName="CPU/hp-esx42.corp.turbonomic.com"/>
<Commodity displayName="Mem/hp-esx42.corp.turbonomic.com"/>
...
</ServiceEntity>
```

GET: entities — commodities bought by entity name or ID

```
vmturbo/api/markets/{name_or_uuid}/{type}/{entityname_or_uuid}/resources
[?property=$property]
[&resource=$resource]
[?starttime=$starttime]
[&endtime=$endtime]
```

Get commodity details for a named entity of a specified type.

`{name_or_uuid}`
String

Required

The name or unique ID of the market.

`{type}`
String

Required

The type of entity to get. Can be one of:

- `entities`
- `virtualapplications`
- `applications`
- `containers`
- `virtualmachines`
- `hosts`
- `datastores`
- `diskarrays`
- `storagecontrollers`
- `iomodules`
- `switches`
- `vpods`
- `dpods`
- `virtualdatacenters`
- `datacenters`
- `groups`

`{entityname_or_uuid}`
String

Required

Internal name or ID of the entity. The API returns entries for that entity.

`property`
String

Optional

One of the following entity properties:

- `priceIndex`
- `Produces`
- `ProducesSize`
- `rOI`

`resource`	String Optional The name of a resource bought by the entity.
`starttime`	UTC Time Optional UTC time in ms. The start and end times for a historic range of entries. If you provide no value or an invalid value for `starttime`, this returns the current data. If you provide a value for `starttime`, but not `endtime`, this uses the range from `starttime` to current. If you provide zero for `starttime`, this returns the earliest stored data.
`endtime`	UTC Time Optional UTC time in ms.

EXAMPLE: Get commodities bought by a given datastore:

URL

```
https://administrator:administrator@10.10.172.91/vmturbo/api/markets/Market/datastores/datastore-2904/resources
```

Returns

```
<ServiceEntity>
<Commodity displayName="Extent/datastore4" used="1" utilization="0"/>
<Commodity displayName="StorageAccess/datastore4[DiskArray-datastore4]" used="1" utilization="0.02"/>
<Commodity displayName="StorageLatency/datastore4[DiskArray-datastore4]" used="0" utilization="0"/>
</ServiceEntity>
```

GET: entities — related entities

```
vmturbo/api/markets/{marketname}/{type1}/{entityname_or_uuid}/{type2}
[?property=$property]
[&resource=$resource]
[?starttime=$starttime]
[&endtime=$endtime]
```

Get a list of entities that are related to the specified entity.

The relationships entities you can query are:

entities

Relationship	Related To

Includes	`actionitems`
	`auditentries`
	`notifications`
	`services`
	`resources`

virtualapplications

Relationship	Related To
Hosted By	`applications`
Includes	`services` `resources`

applications

Relationship	Related To
Hosts	`virtualapplications`
Hosted By	`virtualmachines`

Includes	`actionitems`
	`notifications`
	`services`
	`resources`

Layered Over	`containers`

containers

Relationship	Related To
Layered Over	`virtualmachines`
Underlying	`applications`

virtualmachines

Relationship	Related To
Hosts	`applications`
Layered Over	`datastores`
Hosted By	`hosts`

Includes	`actionitems`
	`notifications`
	`services`
	`resources`

Underlying	`containers`

hosts

Relationship	Related To
Hosts	`virtualmachines`
Underlying	`datastores`
Layered Over	`iomodules`
Hosted By	`datacenters`

Includes	`actionitems`
	`notifications`
	`services`
	`resources`

datastores

Relationship	Related To
Layered Over	`hosts`
Hosted By	`diskarrays`

Includes	`actionitems`
	`notifications`
	`services`
	`resources`

diskarrays

Relationship	Related To
Hosts	`datastores`
Hosted By	`storagecontrollers`

Includes	`actionitems`
	`notifications`
	`services`
	`resources`

storagecontrollers

Relationship	Related To
Hosts	`diskarrays`

Includes	`actionitems`
	`notifications`
	`services`
	`resources`

iomodules

Relationship	Related To
Underlying	`hosts`
Layered Over	`switches`
Hosted By	`datacenters`

Includes	`actionitems`
	`notifications`
	`services`
	`resources`

switches

Relationship	Related To
Underlying	`iomodules`

Includes	`actionitems`
	`notifications`
	`services`
	`resources`

vpods

Relationship	Related To
Layered Over	dpods
Underlying	virtualmachines

dpods

Relationship	Related To
Layered Over	hosts datastores
Underlying	vpods

virtualdatacenters

Relationship	Related To
Layered Over	hosts datastores
Hosts	virtualdatacenters
Underlying	virtualmachines
Includes	actionitems notifications services resources

datacenters

Relationship	Related To
Hosts	hosts iomodules
Includes	actionitems notifications services resources

groups

Relationship	Related To
Ordered List of Entities	`hostcomparison` `storagecomparison`
Includes	`actionitems` `notifications`

`{marketname}`

String

Required

The name of the market to query.

`{type1}`

String

Required

The type of entity that is the source of the relationship. Can be one of:

- `entities`
- `virtualapplications`
- `applications`
- `containers`
- `virtualmachines`
- `hosts`
- `datastores`
- `diskarrays`
- `storagecontrollers`
- `iomodules`
- `switches`
- `vpods`
- `dpods`
- `virtualdatacenters`
- `datacenters`
- `datacenters`
- `projections`

`{entityname_or_uuid}`

String

Required

Name or ID of the source entity.

`{type2}`

String

Required

The type of entity that is related to the source entity. Can be one of the listed entity types, but the API only returns data if this is a type that can be related to `type1`. Also, `type2` cannot be the same as `type1`.

`property`	String Optional One of the following entity properties: - `priceIndex` - `Produces` - `ProducesSize`
`service`	String Optional The name of a service provided by the entity.
`resource`	String Optional The name of a resource on the entity. For example, `CPU_utilization`. The returned data excludes all other resources but the named resource. The value string can OR resources together — for example `resource=Mem_utilization|CPU_utilization|IOThroughput_utilization`.
`starttime`	UTC Time Optional UTC time in ms. The start and end times for a historic range of entries. If you provide no value or an invalid value for `starttime`, this returns the current data. If you provide a value for `starttime`, but not `endtime`, this uses the range from `starttime` to current. If you provide zero for `starttime`, this returns the earliest stored data.
`endtime`	UTC Time Optional UTC time in ms.

EXAMPLE: Get the VM that hosts a specified GuestLoad application:

URL

```
https://usr:pwd@10.10.172.89/vmturbo/api/markets/Market/applications/GuestLoad-423fa768-aa0d-11ca-e8a9-d03aa9300547/virtualmachines
```

Returns

```
<ServiceEntities>
<ServiceEntity ApplicationCommodity_utilization="0" Ballooning_utilization="0"
CPU_utilization="0" ClusterCommodity_utilization="0"
DataCenterCommodity_utilization="0" IOThroughput_utilization="0" Mem_utilization="0"
NetThroughput_utilization="0" Q2VCPU_utilization="0" Swapping_utilization="0"
VCPU_utilization="0" VMem_utilization="0" VStorage_utilization="0"
creationClassName="VirtualMachine" displayName="Win2003-Web_Clone-10" name="vm-862"
priceIndex="1" uuid="423fa768-aa0d-11ca-e8a9-d03aa9300547"/>
</ServiceEntities>
```

EXAMPLE: Get the VMs hosted by a specified physical machine:

URL

```
https://usr:pwd@10.10.172.88/vmturbo/api/markets/Market/hosts/host-396/virtualmachines
```

Returns

```
<ServiceEntities>
<ServiceEntity ApplicationCommodity_utilization="0" Ballooning_utilization="0"
CPU_utilization="2.9" ClusterCommodity_utilization="100"
DSPMAccessCommodity_utilization="100" DataCenterCommodity_utilization="100"
DatastoreCommodity_utilization="100" IOThroughput_utilization="0"
Mem_utilization="6.62" NetThroughput_utilization="0" NetworkCommodity_utilization="100"
Q2VCPU_utilization="0.68" StorageAccess_utilization="0"
StorageAmount_utilization="1.02" StorageLatency_utilization="33"
Swapping_utilization="0" VCPU_utilization="2.98" VMem_utilization="0.2"
VStorage_utilization="23.44" creationClassName="VirtualMachine" displayName="SUSE-2"
name="vm-491" priceIndex="1.07" uuid="423f0e2c-cdf8-8727-0650-0a723ae634da"/>
<ServiceEntity ApplicationCommodity_utilization="0" Ballooning_utilization="0"
CPU_utilization="0" ClusterCommodity_utilization="100"
DSPMAccessCommodity_utilization="100" DataCenterCommodity_utilization="100"
DatastoreCommodity_utilization="100" IOThroughput_utilization="0" Mem_utilization="0"
NetThroughput_utilization="0" NetworkCommodity_utilization="100" Q1VCPU_utilization="0"
StorageAccess_utilization="0" StorageAmount_utilization="1.22"
StorageLatency_utilization="0" Swapping_utilization="0" VCPU_utilization="0"
VMem_utilization="0" creationClassName="VirtualMachine" displayName="Win2003-Web_Clone-
6" name="vm-853" priceIndex="1" uuid="423f49e1-4c5d-9eb5-4b5a-210107b60779"/>
<ServiceEntity ApplicationCommodity_utilization="0" Ballooning_utilization="0"
CPU_utilization="0.58" ClusterCommodity_utilization="100"
DSPMAccessCommodity_utilization="100" DataCenterCommodity_utilization="100"
DatastoreCommodity_utilization="100" IOThroughput_utilization="0"
Mem_utilization="0.01" NetThroughput_utilization="0" NetworkCommodity_utilization="100"
Q4VCPU_utilization="0.47" StorageAccess_utilization="0"
StorageAmount_utilization="7.62" StorageLatency_utilization="0"
Swapping_utilization="0" VCPU_utilization="0.77" VMem_utilization="0"
creationClassName="VirtualMachine" displayName="vm-001" name="vm-925" priceIndex="1.02"
uuid="423f99ea-ecd8-1018-f20e-90b3dd7659fe"/>
</ServiceEntities>
```

GET: entities — related notifications

```
vmturbo/api/markets/{marketname}/{type}/{entityname_or_uuid}/notifications
[?category=$category]
[&active=$active]
```

Get a list of notifications that are related to the specified entity. Notifications indicate a problem the appliance has in discovery, or a risk/opportunity that is associated with an action.

`{marketname}`	String Required The name of the market to query.

`{type}`	String Required The type of entity to get. Can be one of: - `virtualapplications` - `applications` - `containers` - `virtualmachines` - `hosts` - `datastores` - `diskarrays` - `storagecontrollers` - `iomodules` - `switches` - `vpods` - `dpods` - `virtualdatacenters` - `datacenters`
`{entityname_or_uuid}`	String Required Internal name or ID of the source entity.
`category`	String — Regular Expression Optional The notification category — either `Discovery` for appliance problems during discovery, or `MarketProblem` for a risk/opportunity associated with a recommended action.
`active`	Boolean Required One of `true` or `false`. If you specify this parameter, the API filters the returned data to include active or inactive notifications.

EXAMPLE: Get the Discovery notifications for a specified host:

URL

```
https://usr:pwd@10.10.172.88/vmturbo/api/markets/Market/hosts/vm-1186/
notifications?category=Discovery
```

Returns

```
<Notifications>
<Notification category="Discovery" clearTime="1334169183621" count="1" description=""
displayName="displayName" duration="266" event="VMware tools not installed"
importance="0" lastNotified="1334153248685" name="VirtualMachine::vm-1186::VMware tools
not installed" notificationObjectClassName="VirtualMachine"
notificationObjectDisplayName="Saba-test-1" notificationObjectUuid="423f3bd9-4668-1036-
f5fe-07be51d4d8cf" severity="Normal" state="NOTIFY" subCategory="Discovery"
uuid="_rCSV0IPfEeGZV470sBfzSg"/>
</Notifications>
```

GET: entities — related actions

`vmturbo/api/markets/{marketname}/{type}/{entityname_or_uuid}/actionitems [?complete=$complete]`

For the specified entity, get a list of recommended actions and the associated risks/opportunities.

`{marketname}`	String Required The name of the market to query.
`{type}`	String Required The type of entity to get. Can be one of: - `virtualapplications` - `applications` - `containers` - `virtualmachines` - `hosts` - `datastores` - `diskarrays` - `storagecontrollers` - `iomodules` - `switches` - `vpods` - `dpods` - `virtualdatacenters` - `datacenters`
`{entityname_or_uuid}`	String Required Internal name or ID of the source entity.
`complete`	Boolean Optional One of `true` or `false`. If you specify this parameter, it filters the returned data to include completed (`true`) or uncompleted (`false`) actions.

EXAMPLE: Get the action items for a specified VM:

URL

`https://usr:pwd@10.10.172.88/vmturbo/api/markets/Market/virtualmachines/vm-427/actionitems`

Returns

```
<ActionItems allowFixing="true" categoryList="Performance Assurance">
<ActionItem acceptTime="" actionType="MOVE" category="Performance Assurance"
creationClassName="ActionItem" current="hp-esx26.corp.turbonomic.com"
currentClassName="PhysicalMachine" description="" displayName="ActionItem/ActionLog/
Market" explanation="Move Virtual Machine "Fedora-2" from Physical Machine "hp-
esx26.corp.turbonomic.com" to Physical Machine "hp-esx28.corp.turbonomic.com", to
prevent CPU congestion in "hp-esx26.corp.turbonomic.com"" importance="13809.99"
name="ACTIONITEM-_tIChaIPfEeGZV470sBfzSg" new="hp-esx28.corp.turbonomic.com"
newClassName="PhysicalMachine" problem="CPU congestion on Physical Machine "hp-
esx26.corp.turbonomic.com"" problemDescription="CPU congestion on Physical Machine "hp-
esx26.corp.turbonomic.com"" progress="0" recommendTime="1334167672911" savings="0.0"
shortDescription="Move Virtual Machine" state="Pending Accept" target="Fedora-2"
targetClassName="VirtualMachine" uuid="_Qaam4IQBEeGZV470sBfzSg">
<Notification category="MarketProblem" description="CPU congestion on Physical Machine
"hp-esx26.corp.turbonomic.com"" displayName="displayName" event="CPU Congestion"
importance="13809.99" name="PhysicalMachine::host-482::CPU Congestion"
notificationObjectClassName="PhysicalMachine" notificationObjectDisplayName="hp-
esx26.corp.turbonomic.com" notificationObjectUuid="Virtual_ESX_4238fbf9-6327-8fac-80d5-
0bed6e16781a" severity="Critical" state="NOTIFY" subCategory="Performance Assurance"
uuid="_QabN8YQBEeGZV470sBfzSg"/>
</ActionItem>
</ActionItems>
```

GET: entities — audit logs for entity

```
vmturbo/api/markets/{marketname}/entities/{uuid}/auditentries
[?starttime=$starttime]
[&endtime=$endtime]
```

Get the audit log entries for the named entity.

`{marketname}`	String Required The name of the market to query.
`{uuid}`	String Required ID of the source entity.
`starttime`	UTC Time Optional UTC time in ms. The start and end times for a historic range of entries. If you provide no value or an invalid value for `starttime`, this returns the current data. If you provide a value for `starttime`, but not `endtime`, this uses the range from `starttime` to current. If you provide zero for `starttime`, this returns the earliest stored data.
`endtime`	UTC Time Optional UTC time in ms.

EXAMPLE: Get the audit log for a specific VM:

URL

```
https://cud:cud@10.10.172.88/vmturbo/api/markets/Market/entities/6F86454B-1BBD-476B-9144-78499691E71C/auditentries
```

Returns

```
<audit-log-entries> <audit-log-entry>
<created-at>1328630246986</created-at>
<action-name>NEW_SERVICE_ENTITY</action-name>
<category>Discovery</category>
<user-name>SYSTEM</user-name>
<target-object-class>VirtualMachine</target-object-class>
<target-object-name>6F86454B-1BBD-476B-9144-78499691E71C</target-object-name>
<target-object-uuid>6F86454B-1BBD-476B-9144-78499691E71C</target-object-uuid>
...
</audit-log-entry> ... </audit-log-entries>
```

POST: entities — add entity copies to plan

```
vmturbo/api/markets/{marketname}/entities/{entityname_or_uuid}
--data count=[int]&budget=[float]&constraint=[enabled|disabled]&templateName=[String]
```

Add copies of an entity to a planner market. For example, to increase the workload in a plan, you can add more VMs to it. Or you can add physical machines or datastores to increase the planned capacity.

`{marketname}`	String Required The internal name of the market you will modify.
`{entityname_or_uuid}`	String Required The internal name or unique ID of the entity you're copying to add to the plan. If the identified entity exists, the plan adds this copy.
`count`	Integer Required The number of entities to add. If you don't specify a count, the API defaults to zero, and will not add any copies to the plan.
`budget`	Float Required The priority this application has to purchase commodities from the hosting VM. Express this as a value between 0 and 1, where 1 is 100%. If you don't specify a value, the API defaults to zero.
`constraint`	String Optional Whether to enable processing on this market. Can be one of `enabled` or `disabled`. Default is `enabled`.

`templateName` String

Optional

The name of a template to copy. If this is present and there is no match for the `entityname_or_uuid`, this adds a copy of the template to the plan.

EXAMPLE: Add two VMs to the workload of a plan:

URL

```
https://usr:pwd@10.10.172.88/vmturbo/api/markets/_VWg_wHgUEeGGYt0yqL27hQ_BasePlan/
entities/_YAvhOYTUEeGRw_gjztEB3Q?count=2
```

POST: entities — set host state

```
vmturbo/api/markets/{marketname}/hosts/{entityname_or_uuid}/state/{state_val}
--data count=[int]&budget=[float]&constraint=[enabled|disabled]&templateName=[String]
```

For a host in a planner market, set the state of that host. This setting only affects the data in the market — You cannot use this call to change the states of a host in the real-time market.

`{marketname}` String

Required

The internal name of the market you will modify.

`{entityname_or_uuid}` String

Required

The internal name or unique ID of the host to modify.

`{state_val}` String

Required

The state you will set this host to. Can be one of:

- `maintenanceOn`
- `maintenanceOff`
- `powerOn`
- `powerOff`
- `failoverOn`
- `failoverOff`

EXAMPLE: Put the indicated host in maintenance mode:

URL

```
https://usr:pwd@10.10.172.88/vmturbo/api/markets/_VWg_wHgUEeGGYt0yqL27hQ_BasePlan/
hosts/_YAvhOYTUEeGRw_gjztEB3Q/state/maintenanceOn
```

POST: entities — specify placement policies in a plan

`vmturbo/api/markets/{marketname}/{entity_type1}/{name_or_uuid}/{services_or_resources}/{entity_type2}/key/{name_or_uuid2}`

Specify access commodities an entity can provide or consume as a way to specify workload placement. This is analogous to specifying workload placement policies in the Turbonomic user interface.

An access commodity specifies the `services` that an entity provides, or the `resources` an entity consumes. To establish workload placement, you set up a bi-directional relationship between service providers and resource consumers.

For example, assume you create `services` access commodities for Host_A and Host_B. For each host, you specify a different set of `datastore`, `network`, and `datacenter` services the host will provide. Then for a given VM, you can specify a `resources` access commodity for Host_A or Host_B. The choice of host determines which set of resources the VM can use.

Other examples include:

- Limit a set of VMs to a single host: Specify `services` access commodities for that host to provide the necessary services, then specify `resources` access commodities for each VM to consume from that host. For example, you could set the plan's scope to a single cluster, and assign specific VMs in that scope to a single host.
- Limit a set of VMs to two specific hosts, but one datastore: Specify two `services` commodities for the given datastore to provide both hosts as services. Then specify `resources` access commodities for each VM to consume from that datastore.

`{marketname}`	String Required The internal name of the market you will modify.
`{entity_type1}`	String Required For a `services` access commodity, can be one of `hosts` or `datastores`. For a `resources` access commodity, must be `virtualmachines`.
`{name_or_uuid}`	String Required The internal name or uuid of the type-1 entity.
`{services_or_resources}`	String Required The type of access commodity to create. Can be `services` or `resources`.
`{entity_type2}`	String Required Can be one of `host`, `datastore`, `datacenter`, `network`, or `cluster`. This value cannot be the same as the value for `{entity_type1}`.
`{name_or_uuid2}`	String Required The internal name of the type-2 entity.

EXAMPLE: Specify a datastore as a service provided by a given host:

URL

```
https://usr:pwd@10.10.172.86/vmturbo/api/markets/_Y7bfIIi2EeGrq73x6zqyeQ_BasePlan/
hosts/host-30__Y7bfIIi2EeGrq73x6zqyeQ_BasePlan/services/datastore/key/datastore-
1150__Y7bfIIi2EeGrq73x6zqyeQ_BasePlan
```

Returns

```
<TopologyElements><TopologyElement creationClassName="DatastoreCommodity"
displayName="DatastoreCommodity/hp-esx37.corp.turbonomic.com" name="datastore-
1150__Y7bfIIi2EeGrq73x6zqyeQ_BasePlan_name" uuid="_0eUF4Io2EeGWyNd0NodRag"/>
</TopologyElements>
```

DELETE: entities — delete entity from a plan by type

`vmturbo/api/markets/{name_or_uuid}/{type}/{entityname_or_uuid}`

Delete a named entity from a plan. For example, you can delete a named host from the plan. In this case, when you run the plan the results will not include that host.

`{name_or_uuid}` String

Required

The name or unique ID of the market.

`{type}` String

Required

The type of entity to delete. Can be one of:

- `entities`
- `virtualapplications`
- `applications`
- `virtualmachines`
- `hosts`
- `datastores`
- `diskarrays`
- `storagecontrollers`
- `iomodules` (for switch entities)
- `virtualdatacenters`
- `datacenters`
- `entitygroups`

`{entityname_or_uuid}` String

Required

Internal name or ID of the entity.

EXAMPLE: Delete a host from the plan:

URL

```
https://usr:pwd@10.10.172.86/vmturbo/api/markets/_Y7bfIIi2EeGrq73x6zqyeQ_BasePlan/
hosts/DELL3__Y7bfIIi2EeGrq73x6zqyeQ_BasePlan
```

Returns

- `200` for success

DELETE: entities — delete access commodities from a plan

`vturbo/api/markets/{marketname}/{entity_type1}/{name_or_uuid}/{services_or_resources}/{entity_type2}/key/{name_or_uuid2}`

To remove the constraints set by access commodities, you delete them from a planner market. For example, assume you set `resources` access commodities that limit a number of VMs to run on a given host. You can use a DELETE request to remove that constraint from each VM individually. To identify an existing access commodity, the URL specifies the affected entities — you do not identify the access commodity by its name or uuid.

`{marketname}`	String Required The internal name of the market you will modify.
`{entity_type1}`	String Required For a `services` access commodity, can be one of `hosts` or `datastores`. For a `resources` access commodity, must be `virtualmachines`.
`{name_or_uuid}`	String Required The internal name or uuid of the type-1 entity.
`{services_or_resources}`	String Required The type of access commodity to create. Can be `services` or `resources`.
`{entity_type2}`	String Required Can be one of `host`, `datastore`, `datacenter`, `network`, or `cluster`. This value cannot be the same as the value for `{entity_type1}`.
`{name_or_uuid2}`	String Required The internal name of the type-2 entity.

EXAMPLE: Delete an access commodity:

URL

```
https://usr:pwd@10.10.172.86/vmturbo/api/markets/_Y7bfIIi2EeGrq73x6zqyeQ_BasePlan/
virtualmachines/E1013BDE-C9C4-4659-902E-A928218CCFC2__Y7bfIIi2EeGrq73x6zqyeQ_BasePlan/
resources/host/key/DELL4__Y7bfIIi2EeGrq73x6zqyeQ_BasePlan
```

Returns

- `200` for success

reservations

The `reservations` resource returns the list of reservations that are currently active on the Turbonomic server. You can also pass the `uuid` of a specific reservation to get a listing of the proposed VM deployments for that item.

With this resource you can:

- Get a list of current reservations
- Create reservations
- Perform an action on a reservation (deploy or retry, for example)
- Delete a reservation

GET: reservations

```
vmturbo/api/reservations/[{uuid}]
--data state=$reserved | $pending
```

Get a list of current reservations. If you provide a `uuid`, returns a list of the VMs that are in the specified reservation.

`{uuid}`

String

Optional

The UUID that identifies the reservation item. When you start a deployment action, the API returns this key on success. If you call the `reservations` resource without a UUID value, the API returns a list of reservation items. If you call the resource with a UUID value, the API returns the reserved VMs.

`state`

String

Optional

If you provide this parameter, filters the response by state. Can be one of:

- `UNFULFILLED`
- `IN_PROGRESS`
- `RESERVED`
- `PENDING`
- `LOADING`
- `DEPLOYING`
- `PLACEMENT_SUCCEEDED`
- `PLACEMENT_FAILED`
- `DEPLOY_SUCCEEDED`
- `DEPLOY_FAILED`
- `RETRYING`
- `FUTURE`

EXAMPLE: Get the reservations proposals currently on the server:

URL

```
https://usr:pwd@10.10.172.86/vmturbo/api/reservations
```

Returns

```
<TopologyElements>
<TopologyElement catalogItem="__gui__Citrix Demonstration Linux Virtual Machine (5)"
creationClassName="Reservation" displayName="Cud_Reservation1" expirationDate="Fri Sep
26 04:00:00 UTC 2014" name="Cud_Reservation1" profileName="TMP-hp-
xen5.corp.turbonomic.com-f4475021-7009-3460-e6fc-01a13deac5a0" startDate="Fri Sep 26
04:00:00 UTC 2014" status="RESERVED" uuid="_RC59UDRGEeSvatd2DB2axg" vmCount="5"
vmPrefix="CudResVM"/>
...
</TopologyElements>
```

EXAMPLE: Get details of a specific deployment proposal:

URL

```
https://usr:pwd@10.10.172.86/vmturbo/api/deployitems/_c4cZ0R1MEeKvyvBZ34hwqQ
```

Returns

```
<VirtualMachines>
<VirtualMachine datastore="QS3HVDS2" host="hp-dl593.corp.turbonomic.com"
name="Cud_Reservation2_CudRes2VM_C0"/>
<VirtualMachine datastore="QS3HVDS2" host="hp-dl593.corp.turbonomic.com"
name="Cud_Reservation2_CudRes2VM_C1"/>
</VirtualMachines>
```

POST: reservations — create reservation

Create a reservation.

NOTE: The server does not post the reservation immediately. It can take a few moments for the server to calculate the correct placement.

```
vmturbo/api/reservations
--data reservationName=$string&count=$int&templateName=$string_or_uuid
&deploymentProfile=$string_or_uuid&deployDate=$simple_date&reservationDate=$simple_date
&segmentationUuid[]=$uuid
```

`reservationName`	String Required The name that appears in the user interface for this reservation.
`count`	Integer Required The number of VMs to create in this reservation.

`templateName`

String

Required

The name or UUID of the VM template to use.

`deploymentProfile`

String

Optional

The name or UUID of the deployment profile to use.

For a reservation that uses a discovered template, you should provide the matching discovered deployment profile.

Note that deployment profiles can limit the reservation to a specific scope. You can also limit the reservation via the `segmentationUuid` parameter.

Also note that if you want Turbonomic to deploy the reservation, then you must provide a deployment profile. However, if you use the reservation to calculate the deployment information that you will pass to a 3rd-party orchestration system, then you do not need to provide a deployment profile.

`deployDate`

Simple Date

Optional

`yyyy-MM-dd hh:mm:ss` — The date on which to execute the reservation. For example, `2015-01-10 15:00:00`.

- If you provide today's date, the API ignores the time and deploys the VMs immediately
- If you do not provide a date, the API sets the deploy date to six months from now
- A past date is an error, and the reservation is not created

`reservationDate`

Simple Date

Optional

`yyyy-MM-dd hh:mm:ss` — The date on which to calculate placement and reserve the resources. For example, `2015-01-10 15:00:00`.

- If you provide today's date, the API ignores the time and creates the reservation immediately
- If you do not provide a date, the API creates the reservation immediately
- A past date is an error, and the reservation is not created

`segmentationUuid[]`

uuid array

Optional

An array of unique IDs for the placement policies or groups that constrain scope when deploying this reservation. You can provide UUIDs for entities of these types:

- Cluster
- Datacenter
- Virtual datacenter
- Network
- Placement policies (to get the defined placement policies, see GET: groups — get workload placement policies on page 45)

EXAMPLE: Create a reservation:

URL

```
https://usr:pwd@10.10.172.86/vmturbo/api/reservations/
?reservationName=myRes&vmPrevix=myResVMs&count=5&templateName=VirtualMachine::ibm.corp.
turbonomic.com::TMP-vmTemplateNew2&deploymentProfile=7b8202c5-2d92-414a-a3c7-
b219b61dccb5&action=reserve&deployDate=2015-01-31&reservationDate=2014-10-10 01:00:00
```

Returns

Reservation UUID on success

On failure, returns an error message. Note that the call returns an error if you provide invalid UUID values for `segmentationUiid`.

POST: reservations — execute action

Execute a reservation action.

```
vmturbo/api/reservations/{uuid}
--data action=$string
```

`{uuid}`	String Required The UUID that identifies the reservation item.
`action`	String Required The action to perform with this reservation. Can be one of: - `reserve` - `deploy` - `retry`

EXAMPLE: Retry reservation that is in the pending state:

URL

```
https://usr:pwd@10.10.172.86/vmturbo/api/reservations/
_Kz3OEDR7EeSvatd2DB2axg?action=retry
```

Returns

`true` on success, or `false` on failure

DELETE: reservations

`vmturbo/api/reservations/{uuid}`

Delete the indicated reservation.

`{uuid}`	String required The UUID that identifies the reservation to delete.

EXAMPLE: Delete a reservation:

URL

`https://usr:pwd@10.10.172.86/vmturbo/api/reservations/_m5RAETRbEeSvatd2DB2axg`

Returns

`true` on success, or `false` on failure

reservationstate

The `reservationstate` resource lists the supported reservation states.

GET: reservationstate

`vmturbo/api/reservationstate`

Get a list of supported reservation states.

EXAMPLE: Get the deployment proposals currently on the server:

URL

`https://usr:pwd@10.10.172.86/vmturbo/api/reservationsate`

Returns

```
<ReservationStates>
<ReservationState name="IN_PROGRESS" />
<ReservationState name="RESERVED" />
<ReservationState name="PENDING" />
<ReservationState name="LOADING" />
<ReservationState name="DEPLOYING" />
<ReservationState name="PLACEMENT_SUCCEEDED" />
<ReservationState name="PLACEMENT_FAILED" />
<ReservationState name="DEPLOY_SUCCEEDED" />
<ReservationState name="DEPLOY_FAILED" />
<ReservationState name="RETRYING" /> </ReservationStates>
```

External Applications

You can use the REST API to manage two types of external applications:

- Applications you discover and manage
  By default Turbonomic discovers applications that are running in your environment (see Application Discovery in the Turbonomic Policies view). However, you can use your own external processes to discover and monitor applications in the environment. You can use the REST API to register these applications to appear in the Turbonomic GUI, but set their charted values (resource utilization and services provided) with data from your external process. In this way, you can integrate your external application monitoring with Turbonomic. For more information, see Externally Discovered Applications on page 14.
- Action Manager job applications
  Another use for this resource is to create specific jobs to be run by the Turbonomic Actions Manager.

NOTE: To use the Actions Manager, you must have the VDI Control Module and the Storage Control module enabled for your Turbonomic installation.

For information and examples, see Jobs Executed by the Actions Manager on page 15.

Use application POST methods to:

- Create external applications in the real-time market
- Set application state (external applications should always be `not_monitored`)
- Set application service values
- Set application resource values
- Create Action Manager jobs
- Set capacity for Action Manager jobs

DELETE requests remove the specified application from the market.

NOTE: You should only delete external applications from the real-time market. do not delete applications that are discovered and monitored by Turbonomic from the real-time market. With planner markets, you can delete any applications you want. For more information about the different types of markets, see markets on page 58.

POST: applications — create external application on a VM

`vmturbo/api/markets/{mktname_or_uuid}/virtualmachines/{vm_name_or_uuid}/applications/{app_name}`

Create an application in the market.

`{mktname_or_uuid}`	String Required The market's name or unique identifier.
`{vm_name_or_uuid}`	String Required Provide the name or UUID of the VM that hosts this application.

`{app_name}`	String Required The name of the application you want to create. This serves as a base for the display name and the internal name of the application.

EXAMPLE: Create an application to run on a specified VM:

URL

```
https://guest:guest@123.123.123.12/vmturbo/api/markets/Market/virtualmachines/421ea068-f1f7-3df4-a9b0-6ed258dc515f/applications/externalApp
```

Returns

```
<TopologyElements>
<TopologyElements>
<TopologyElement creationClassName="Application" displayName="externalApp [SUSE-2.1]"
name="externalApp-421ea068-f1f7-3df4-a9b0-6ed258dc515f"
uuid="6504b5ef9021af4c4d3d6e23fb9edb6c4cfe0415"/>
</TopologyElements> SUCCESS
```

In the returned data, note that the application's display name uses the passed name plus the VM's name, and the internal name uses the passed name plus the VM's UUID.

POST: application — set application state: `active` or `not_monitored`

```
vmturbo/api/markets/{mktname_or_uuid}/applications/{app_name_or_uuid}/state/{state_value}
```

Set the application's state. For all external applications that you create on the market this way, you must set the state to `not_monitored`. That frees you to perform the monitoring with an external process, and use the API to set service and resource values accordingly in the market.

`{mktname_or_uuid}`	String Required The market's name or unique identifier.
`{app_name_or_uuid}`	String Required The name or identifier of the application you want to modify.
`{state_value}`	String Required Can be one of `active` or `not_monitored`. For an external application you have added to the market, the state should always be `not_monitored`.

EXAMPLE: Set the state to not_monitored:

URL

```
https://guest:guest@123.123.123.12/vmturbo/api/applications/
da5add10d20cbf67566d1c093150a249f8a40b6e/state/not_monitored
```

Returns

The HTTP result.

POST: applications — specify resource values

`vmturbo/api/markets/{mktname_or_uuid}/applications/{app_name_or_uuid}/resources/{resource_name}/{attribute}/{value}`

Set a value for one of the application resources. To see the resources available on the application, execute a GET for the application resources (`.../market/{marketname_or_uuid}/{app_name_or_uuid}/resources`)

NOTE: If you want the external application to participate in SLA calculations, use this call to set utilization of resources that figure into SLA calculations — For example, you can set utilization of the `SLACommodity` resource.

`{mktname_or_uuid}`	String Required The market's name or unique identifier.
`{app_name_or_uuid}`	String Required The name or identifier of the application you want to modify.
`{resource_name}`	String Required The name of the resource you want to set. For SLA resources, the resource name can be: - `SLACommodity` - `VCPU` - `VMem`
`{attribute}`	String Required The use of the resource. For example: - `capacity` - `peak` - `used` - `utilization`
`{value}`	String Required The value you want to set for the resource.

EXAMPLE: Set VMem utilization to 75% for an application:

URL

```
https://guest:guest@123.123.123.12/vmturbo/api/applications/
da5add10d20cbf67566d1c093150a249f8a40b6e/resources/VMem/utilization/75
```

Returns

The HTTP result.

POST: applications — specify application services

```
vmturbo/api/markets/{mktname_or_uuid}/applications/{app_name_or_uuid}/services/
{service_name}/{attribute}/{value}
```

Set a value for one of the application services. To see the services available on the application, execute a GET for the application services (`.../market/{marketname_or_uuid}/{app_name_or_uuid}/services`)

`{mktname_or_uuid}`	String Required The market's name or unique identifier.
`{app_name_or_uuid}`	String Required The name or identifier of the application you want to modify. This serves as a base for the display name and the internal name of the application.
`{service_name}`	String Required The name of the service you want to set.
`{attribute}`	String Required The aspect of this service that you want to set.
`{value}`	String Required The value you want to set to this service.

EXAMPLE: Set the capacity for VMem utilization for an application:

URL

```
https://guest:guest@123.123.123.12/vmturbo/api/applications/
da5add10d20cbf67566d1c093150a249f8a40b6e/services/VMem/utilization/75
```

Returns

The HTTP result.

POST: applications — create action manager job on a VM

`vmturbo/api/markets/Market/virtualmachines/{vm_name_or_uuid}/applications/{app_name}/resources/ActionPermit/used/{value}`

Create a job to be handled by the Action Manager for a specific VM.

`{vm_name_or_uuid}`	String Required The VM this job will run on.
`{app_name}`	String Required The name of the Action Manager job to run.
`{value}`	Int Required The number of Action Manager tokens this job uses.

EXAMPLE: Set Windows Security Patch job in the given VM:

URL

`https://guest:guest@123.123.123.12/vmturbo/api/markets/Market/virtualmachines/423f3880-ca50-5e4c-c819-3f099203ed1e/applications/Windows_Security_Patch_2/resources/ActionPermit/used/3`

Returns

The HTTP result.

POST: applications — create action manager job on a group of VMs

`vmturbo/api/groups/{group_uuid}/applications/{app_name}/resources/ActionPermit/used/{value}`

Create a job to be handled by the Action Manager for all the VMs in a group.

`{group_uuid}`	String Required The VM this job will run on.
`{app_name}`	String Required The name of the Action Manager job to run.
`{value}`	Int Required The number of Action Manager tokens this job uses, per VM.

EXAMPLE: Create the virus scan job for all VMs in a group:

URL

```
https://guest:guest@123.123.123.12/vmturbo/api/groups/
f4faebb02daa76ba4190e04767407a024ef3856e/applications/virusscan/resources/ActionPermit/
used/1
```

Returns

The HTTP result.

POST: applications — set action manager capacity on an individual storage controller

```
vmturbo/api/markets/Market/storagecontrollers/{name_or_uuid}/resources/ActionPermit/
capacity/{value}
```

The action manager execution is associated with a storage controller. Use this call to set the number of action manager tokens this storage controller can issue.

`{name_or_uuid}`	String Required The storage controller that will have this capacity.
`{value}`	Int Required The number of Action Manager tokens this storage controller will issue.

EXAMPLE: Set the capacity for job tokens on a storage controller:

URL

```
https://guest:guest@123.123.123.12/vmturbo/api/markets/Market/storagecontrollers/
6e946926b47d4ee6620704e70d6b126eeba83a5d/resources/ActionPermit/capacity/500
```

Returns

The HTTP result.

POST: applications — set action manager capacity on a group of storage controllers

`vmturbo/api/groups/storagecontrollers/{group_uuid}/resources/ActionPermit/capacity/{value}`

The action manager execution is associated with a storage controller. Use this call to set the number of action manager tokens each storage controller can issue.

`{group_uuid}`	String Required The group of storage controllers that will get this capacity for each member.
`{value}`	Int Required The number of Action Manager tokens each storage controller will issue.

EXAMPLE: Set the capacity for job tokens on all storage controllers in the group:

URL

`https://guest:guest@123.123.123.12/vmturbo/api/groups/_S4k98yFQEd-S4dn3K4_SSD/resources/ActionPermit/capacity/600`

Returns

The HTTP result.

DELETE: applications — remove external application

`vmturbo/api/markets/{mktname_or_uuid}/applications/{app_uuid}`

Delete the application from the market. You can also use this to remove an action manager job from a VM.

`{mktname_or_uuid}`	String Required The market's name or unique identifier.
`{app_uuid}`	String Required The name or identifier of the application you want to modify. This serves as a base for the display name and the internal name of the application.

EXAMPLE: Remove the identified external application:

URL

```
https://guest:guest@123.123.123.12/vmturbo/api/markets/Market/applications/
da5add10d20cbf67566d1c093150a249f8a40b6e
```

Returns

The HTTP result.

DELETE: applications — remove action manager job from a group of VMs

`vmturbo/api/groups/{group_uuid}/applications/{app_name}`

Delete the action manager job from a group of VMs.

`{group_uuid}`	String Required The group's unique identifier.
`{app_name}`	String Required The name of the application you want to remove.

EXAMPLE: Remove the virus scan job from a group of VMs:

URL

```
https://guest:guest@123.123.123.12/vmturbo/api/groups/
f4faebb02daa76ba4190e04767407a024ef3856e/applications/virusscan
```

Returns

The HTTP result.

targets

The `targets` resource provides access to the targets that are set up on your appliance. You can use it to:

- Get a list of current targets
- Trigger discovery on the existing targets
- Configure new targets
- Delete targets

NOTE: For more information about targets, please be sure to review the Turbonomic online documentation. You should not modify targets without a full understanding f how they work with Turbonomic.

GET: targets

`vmturbo/api/targets/[{name_or_uuid}]`

Get a list of targets on the appliance. Optionally, you can get the entry for a specific target.

`{name_or_uuid}`	String
	Optional
	The name of the target, or its unique identifier.

EXAMPLE: Get the list of targets on the appliance:

URL

`https://user:pwd@10.10.172.88/vmturbo/api/targets`

Returns

```
<TopologyElements>
<TopologyElement creationClassName="VMTTarget" displayName="vsphere-
dc5.corp.turbonomic.com" name="vsphere-dc5.corp.turbonomic.com" targetType="vCenter"
uuid="_LBQxEO16EeK2YMcop6_HIA"/>
<TopologyElement creationClassName="VMTTarget" displayName="10.10.172.203"
name="10.10.172.203" targetType="vCenter" uuid="_C1ZFkOPnEeKb3IpiiryX_Q"/>
<TopologyElement creationClassName="VMTTarget" displayName="10.10.128.165"
name="10.10.128.165" targetType="XenServer" uuid="_yh_6IOPnEeKb3IpiiryX_Q"/>
<TopologyElement creationClassName="VMTTarget" displayName="10.10.172.206"
name="10.10.172.206" targetType="vCenter" uuid="_ghGbEOl6EeKmRKv4vQP8JA"/>
<TopologyElement creationClassName="VMTTarget" displayName="10.80.34.89"
name="10.80.34.89" targetType="XenServer" uuid="_3RNbQOPnEeKb3IpiiryX_Q"/>
</TopologyElements>
```

POST: targets — rediscover existing targets

```
vmturbo/api/targets/[{name_or_uuid}]
```

Run discovery on all targets on the appliance. Optionally, provide a target name or UUID to run discovery on just that target.

NOTE: To see the results of discovery, execute a GET request on the discoveryprogress on page 114resource.

`{name_or_uuid}` String

Optional

The name of the target, or its unique identifier. If you provide a value, this call executes rediscovery on that target.

EXAMPLE: Run discovery on all targets on the appliance:

URL

```
https://guest:guest@123.123.123.12/vmturbo/api/targets
```

Returns

The HTTP result.

POST: targets — configure a new target or reconfigure an existing target

```
vmturbo/api/targets/[{name_or_uuid}]
--data targetType=$type&nameOrAddress=$address&username=$user&password=$password
&username2=$user2&password2=$password2&domainName=$domain
```

This call adds a new target to the appliance. Optionally, provide a target name or UUID to update an existing target's configuration.

NOTE: After adding a target, you must execute rediscovery. Before adding targets to the appliance, you should understand target configuration via the GUI. For more information about targets, please be sure to review the Turbonomic online documentation. You should not modify targets without a full understanding of how they work with Turbonomic.

For the passed data, you do not need to provide all attribute/value pairs. Some target types require different data. See the Turbonomic online documentation for information about the data required for each target type.

`{name_or_uuid}` String

Optional

The name of the target, or its unique identifier. Provide this to reconfigure an existing target. If you do not provide a name or UUID, this call creates a new target.

`targetType`

String

Required

The type of target to add. Can be one of:

- `vCenter`
- `Hyper-V`
- `XenServer`
- `RHEV`
- `NetScaler`
- `vCloudDirector`
- `OpenStack`
- `CloudStack`
- `VMM`
- `UCS`
- `NetApp`
- `VNX`
- `HP3PAR`
- `PureStorage`
- `Nutanix`
- `AWS`
- `Azure`
- `sFlow`
- `NetFlow`
- `Arista`
- `WebSphere`
- `WebLogic`
- `JBoss`
- `Tomcat`
- `Oracle`
- `SQLServer`
- `MySQL`
- `MSExchange`
- `VMTAppliance`

`nameOrAddress`

String

Required

The IP address or host name of the target server.

`username`

String

Required

The username for an account that gives Turbonomic access to the target server.

`password`

String

Required

The password for an account that gives Turbonomic access to the target server.

`username2`	String Optional For `vCloudDirector` targets, the username to log onto the vCenter Servers managed by the target.
`password2`	String Optional For `vCloudDirector` targets, the password to log onto the vCenter Servers managed by the target.
`domainName`	String Optional For Hyper-V targets, the full domain name of the cluster managed by that target.
`port`	String Optional The port on the target that Turbonomic can use to connect.
`scopeName`	String Optional For targets supported by the Application Control Module (application servers, database servers, and Microsoft applications). For these targets, you can specify the name of a VM group or cluster, and Turbonomic will scan that scope for VMs that match the specified port for this target. Note that if you specify a scope, you must also specify a port.
`id`	String Optional For `VMTAppliance` targets, a string to identify an underlying appliance in the aggregating appliance user interface.

EXAMPLE: Add a new target to the appliance:

URL

```
https://guest:guest@123.123.123.12/vmturbo/api/targets
?targetType=vCenter
&nameOrAddress=10.10.112.202
&username=MyUsername
&password=MyPassword
```

Returns

The HTTP result.

This is the result of posting the request, and does not indicate that you successfully added a new target. To verify that you added the target, get the `targets` resource and verify that the new target is listed in the returned data.

DELETE: targets

`vmturbo/api/targets/{name_or_uuid}`

Delete the specified target from the appliance.

`{name_or_uuid}`	String Required The name of the target, or its unique identifier.

EXAMPLE: Delete a target from the appliance:

URL

`https://guest:guest@123.123.123.12/vmturbo/api/targets/_C1ZFkOPnEeKb3IpiiryX_Q`

Returns

The HTTP result.

This is the result of posting the request, and does not indicate that you successfully added a new target. To verify that you added the target, get the `targets` resource and verify that the new target is listed in the returned data.

externaltargets

The `externaltargets` resource adds a target to Turbonomic that will be discovered and managed by a custom probe.

POST: externaltargets

```
vmturbo/api/externaltargets
--data type=MyCustomTargetType&targetIdentifier=MyIdString
```

Use a custom probe to attach to a target. The actual parameters you pass depend on the definition of the custom probe. Each custom probe defines the required and optional fields for attaching it to a target. Before adding a target with a custom probe, you must know which parameters it requires.

When you use this method to add an external target, if the call fails the returned data includes a sample URL that identifies any missing parameters.

The following list of parameters shows the two fields that are required to add an external target via every custom probe. A custom probe can include other fields that are required or optional, and the custom probe can include string validation for these fields. Before calling this method, you should review the requirements for the given probe.

`type`	String Required The type of target to add, as defined in the probe configuration.
`targetIdentifier`	String Required A unique string that you provide to identify the target within the Turbonomic server. Turbonomic uses this string internally when working with the target.

EXAMPLE: Add a target to be managed by a custom probe:

URL

```
https://guest:guest@123.123.123.12/vmturbo/api/externaltargets
?type=MyType&address=10.10.12.34
```

Returns

On success, `Add target succeed!`

On failure, the following string:

```
Failed to add target, please provide <MissingRequiredField>!
Sample URL: https://username:pasword@yourIpAddress/vmturbo/api/externaltargets?
type=TYPE&targetIdentifier=TARGETID{&yourOptionalFields}
```

Where `<MissingRequiredFields>` is a list of fields that have been marked as `required` in the custom probe.

discoveryprogress

The `discoveryprogress` resource returns the list showing the count of discovered entities on the appliance. This corresponds to the Environment Summary chart on the Admin > Target Configuration view.

GET: discoveryprogress

`vmturbo/api/discoveryprogress`

Get a list of discovered entities on the appliance.

EXAMPLE: Get the discovery progress for the server:

URL

`https://usr:pwd@10.10.172.86/vmturbo/api/discoveryprocess`

Returns

```
<DiscoveryProgressCounts>
<DiscoveryProgressCount className="DataCenter" count="4"/>
<DiscoveryProgressCount className="Storage" count="19"/>
<DiscoveryProgressCount className="Network" count="9"/>
<DiscoveryProgressCount className="VirtualMachine" count="77"/>
<DiscoveryProgressCount className="PhysicalMachine" count="22"/>
</DiscoveryProgressCounts>
```

auditentries

The `auditentry` resource returns the audit log entries that are stored on the Turbonomic server.

GET: auditentries

Get a list of audit log entries. If you provide a time range, it returns the entries generated within that range.

NOTE: This resource returns all the log entries on the server. You can get entries for a specific entity by requesting `auditentries` through the `market` resource. For more information, see GET: entities — audit logs for entity on page 87.

```
vmturbo/api/auditentries
[?starttime=$starttime]
[&endtime=$endtime]
```

`starttime`	UTC Time Optional UTC time in ms. The start and end times for a historic range of entries. If you provide no value or an invalid value for `starttime`, this returns the current data. If you provide a value for `starttime`, but not `endtime`, this uses the range from `starttime` to current. If you provide zero for `starttime`, this returns the earliest stored data.
`endtime`	UTC Time Optional UTC time in ms.

EXAMPLE: Get all the audit log entries stored on the server:

URL

```
https://usr:pwd@10.10.172.86/vmturbo/api/auditentries?starttime=0
```

Returns

```
<audit-log-entries> <audit-log-entry>
<created-at>1328630246986</created-at>
<action-name>NEW_SERVICE_ENTITY</action-name>
<category>Discovery</category>
<user-name>SYSTEM</user-name>
<target-object-class>VirtualMachine</target-object-class>
<target-object-name>6F86454B-1BBD-476B-9144-78499691E71C</target-object-name>
<target-object-uuid>6F86454B-1BBD-476B-9144-78499691E71C</target-object-uuid>
...
</audit-log-entry> ... </audit-log-entries>
```

notifications

The `notifications` resource provides access to notification and problems as they are managed by the appliance. You can use it to:

- Get lists of notifications
- Get related action items

GET: notifications — get list of notifications

```
vmturbo/api/notifications
[?category=$category]
[&active=$active]
[&starttime=$starttime]
[&endtime=$endtime]
```

Get a list of notifications currently in the appliance. You can filter the list by category, or active/inactive state.

`category` String

Optional

The notification category. Can be one of:

- `MarketProblem`
  Issues Turbonomic identifies within your virtual environment.
- `Discovery`
  Issues that occur as Turbonomic performs discovery.
- `Monitoring`
  Issues that affect Operations Manager as it monitors your environment.
- `Control`
  Issues that affect Turbonomic as it performs recommended actions.
- `Mediation`
  Communication issues that arise when Turbonomic sends commands to discover, monitor, or change your environment.
- `Healthcheck`
  Issues that affect Turbonomic performance. These issues are discovered via periodic Turbonomic health check tests.
- `InterAppliance`
  Issues that occur on an aggregating appliance as Turbonomic communicates with target Turbonomic appliances.

`active` Boolean

Optional

One of `true` or `false`. `true` returns active notifications; `false` returns inactive notifications; if you don't provide this parameter, the resource returns all notifications.

`starttime`	UTC Time Optional UTC time in ms. The start and end times for a historic range of entries. If you provide no value or an invalid value for `starttime`, this returns the current data. If you provide a value for `starttime`, but not `endtime`, this uses the range from `starttime` to current. If you provide zero for `starttime`, this returns the earliest stored data.
`endtime`	UTC Time Optional UTC time in ms.

EXAMPLE: Get all the monitoring notifications:

URL

```
https://guest:guest@123.123.123.12/vmturbo/api/notifications?category=Monitoring
```

Returns

```
<Notifications>
<Notification category="Monitoring" clearTime="1313605817811" count="1"
description="Exception while browsing the datastore" displayName="displayName"
duration="208" event="Exception" importance="0" lastNotified="1313593341800"
name="Storage::datastore-654::Exception" notificationObjectClassName="Storage"
notificationObjectDisplayName="swdisk5" notificationObjectUuid="4ceddc81-75debb29-b334-
005056b8000d" severity="MAJOR" state="NOTIFY" subCategory="Monitoring"
uuid="_6K2OgMjhEeCmwpw4r7-jfw"/>
</Notifications>
```

GET: notifications — get related action items

`vmturbo/api/notifications/{name_or_uuid}/actionitems`
`[?complete=$category]`

For `MarketProblem` notifications, get a list of associated action items. You can filter the list by completed/uncompleted state.

`{name_or_uuid}`	String Required The name or unique ID of the notification you want to query
`complete`	Boolean Optional One of `true` or `false`. true returns completed action items; false returns uncompleted action items; if you don't provide this parameter, the resource returns all action items.

EXAMPLE: Get the action items associated with notifications on a specific entity:

URL

```
https://guest:guest@123.123.123.12/vmturbo/api/notifications/_4PsT8cjiEeCmwpw4r7-jfw/
actionitems
```

Returns

```
<ActionItems>
<ActionItem acceptTime="" actionType="MOVE" category="WORKLOAD_BALANCING"
creationClassName="ActionItem" current="hp-esx27.corp.turbonomic.com"
currentClassName="PhysicalMachine" description="" displayName="ActionItem/ActionLog/
Market" explanation="Move Virtual Machine vm-d-03 from Physical Machine hp-
esx27.corp.turbonomic.com to Physical Machine hp-esx26.corp.turbonomic.com, to comply
with Workload Placement::cluster2" importance="20000" name="ACTIONITEM-
_DwasisjgEeCmwpw4r7-jfw" new="hp-esx26.corp.turbonomic.com"
newClassName="PhysicalMachine" problem="Violation of Workload Placement::cluster2 for
vm-d-03" progress="0" recommendTime="1313606358297" shortDescription="Move Virtual
Machine" state="PENDING_ACCEPT" target="vm-d-03" targetClassName="VirtualMachine"
uuid="_4Prs4MjiEeCmwpw4r7-jfw"/>
</ActionItems>
```

actionlogs

The `actionlogs` resource manages To DO lists that are currently on the appliance. The appliance can have multiple To Do lists — one for the real-time market, one for each active plan, and one for Cluster Capacity planning.

You can use this resource to:

- Get a list of the current action logs
- Get lists of action items from a log
- Accept or reject specific action items

GET: actionlogs — list of action logs

`vmturbo/api/actionlogs`

Get a list of action logs (To Do lists) currently in the appliance.

EXAMPLE: Get a list of action logs:

URL

`https://usr:pwd@10.10.172.86/vmturbo/api/actionlogs`

Returns

```
<TopologyElements>
<TopologyElement creationClassName="ActionLog" displayName="ActionLog/Market"
name="Market_ActionLog" uuid="_X1DXsIo8EeGvU88b2rcjMw"/>
<TopologyElement creationClassName="ActionLog" displayName="ActionLog/Market_Default"
name="Market_Default_ActionLog" uuid="_X9or8Yo8EeGvU88b2rcjMw"/>
</TopologyElements>
```

GET: actionlogs — action items in an action log

`vmturbo/api/actionlogs/{log_name_or_uuid}/actionitems`
`[?complete=$boolean]`

Get a list of action log items in the identified action log.

`{log_name_or_uuid}`	String Required The name or ID of the action log you want to query.
`complete`	Boolean Optional When `true`, lists all actions in the To DO list, as though **Show All** was enabled in the GUI. When `false`, lists only the top actions, the same as **Show Top** in the GUI. Because this is an optional parameter, you can simply omit it in any case where you would want to set it to false.

EXAMPLE: Get the action items from a specified log:

URL

```
https://usr:pwd@10.10.172.86/vmturbo/api/actionlogs/_X1DXsIo8EeGvU88b2rcjMw/actionitems
```

POST: actionlogs — accept or reject an action

```
vmturbo/api/actionlogs/{logname_or_uuid}/actionitems/{itemname_or_uuid}
--data action = accept|reject
```

Accept or reject an action item.

`{logname_or_uuid}`	String Required The name or ID of the action log you want to query.
`{itemname_or_uuid}`	String Required The name or ID of the action item you want to modify.
`action`	String Required one of `accept` or `reject` The setting to make on the specified action item.

EXAMPLE: Accept an action:

URL

```
https://usr:pwd@10.10.172.86/vmturbo/api/actionlogs/_X1DXsIo8EeGvU88b2rcjMw/
actionitems/_m6BbYIrvEeGvU88b2rcjMw?action=accept
```

Returns

- `success` for success

inventory

The `inventory` resource gets summaries of inventory per target.

GET: inventory

Get a list of inventory summaries.

`vmturbo/api/inventory`

EXAMPLE: Get inventory summary

URL

`https://guest:guest@123.123.123.12/vmturbo/api/inventory`

Returns

A list of roles:

```
<VMTInventorySummary>
<OpsManagerInventorySummary Datastore="10" Groups="3" OperationsManager="10.10.224.131"
PhysicalMachine="12" TotalObjectCount="41" VirtualMachine="19">
<TargetInventorySummary Target="10.10.128.165">
<GroupInventorySummary Group="PMs_Xen DataCenter\ResourcePool56"
GroupUUID="1432c728ba16589bbbd7ba880b616d4818a76a94" TotalObjectCount="10"
<InventoryInfo Active_Datastore="2" />
<InventoryInfo Active_PhysicalMachine="4" />
<InventoryInfo Active_VirtualMachine="0" />
<InventoryInfo Other_Datastore="4" />
<InventoryInfo Other_PhysicalMachine="0" />
<InventoryInfo Other_VirtualMachine="0" />
</GroupInventorySummary>
</TargetInventorySummary>
<TargetInventorySummary Target="10.10.150.201">
<GroupInventorySummary Group="PMs_DataCenter55\Cluster-2"
GroupUUID="5d8df019a389d6acd67e5a76d875817f306b9d52" TotalObjectCount="19">

...

</GroupInventorySummary>
<GroupInventorySummary Group="PMs_DataCenter55\Cluster-1"
GroupUUID="3fee0381ca889933235058b85669f1d301c4e2f9" TotalObjectCount="14">

...

</GroupInventorySummary>
</TargetInventorySummary>
</OpsManagerInventorySummary>
</VMTInventorySummary>
```

settings

The `settings` resource lists the action modes that you have set up for groups in your environment.

GET: settings

Get a list of inventory summaries.

`vmturbo/api/settings`

EXAMPLE: Get a list of settings

URL

`https://guest:guest@123.123.123.12/vmturbo/api/settings`

Returns

A list of roles:

```
<VMTPolicySummary>
<OpsManagerPolicySummary OperationsManager="10.10.224.131">
<TargetPolicySummary Target="NA">
<GroupPolicySummary Group="PMs_Xen DataCenter\ResourcePool56"
GroupUUID="1432c728ba16589bbbd7ba880b616d4818a76a94">
<PolicyInfo name="PM Provision" setting="RECOMMEND" />
<PolicyInfo name="PM Start" setting="RECOMMEND" />
<PolicyInfo name="PM Suspend" setting="RECOMMEND" />
<PolicyInfo name="PM Terminate" setting="RECOMMEND" />
</GroupPolicySummary>
</TargetPolicySummary>

...

</OpsManagerPolicySummary>
</VMTPolicySummary>
```

schedules

The `schedules` resource manages scheduled action restriction windows.

GET: schedules

`vmturbo/api/schedules/[{uuid}]`

Get a list of scheduled action windows. If you provide `uuid`, returns the entry for that scheduled action window.

`{uuid}`	String Optional The action window's unique ID. Can be a comma-delimited list of ID values (no spaces) to get multiple action windows.

EXAMPLE: Get a list of scheduled action windows

URL

`https://guest:guest@123.123.123.12/vmturbo/api/schedules`

Returns

A list of action windows:

```
<Schedule>
<Policy action="moveVM" actionMode="MANUAL" color="0" endDate="08-11-2015"
endTime="22:00" group="_S4k99yFQEd-S4dn3K4_SSA" name="Move_DAILY"
note="test"recur="FREQ=DAILY;" startDate="04-10-2015" startTime="21:00" uuid="_6dX-
0N_DEeSzLoNDXeGKHg" />
...</Schedule>
```

PUT: schedules

```
vmturbo/api/schedules/{uuid}
--data name=$string&startDate=$SimpleTime&endDate=$SimpleTime&startTime=$SimpleTime&
endTime=$SimpleTime&color=$integer&group=$uuid&action=$actionType&actionMode=$modeType
&recur=$string&note=$string
```

Edit an existing scheduled action window. The data fields are optional — This call modifies the action window with the values you pass, and returns a new UUID.

NOTE: The start date and time must be earlier than the end date and time.

`{uuid}`	UUID Required The action window's unique ID. Can be a comma-delimited list of ID values (no spaces)
`name`	String Optional The action window's name.
`startDate`	Simple Date Optional MM-DD-YYYY — for example, 04-10-2015. The date when the action window opens.
`endDate`	Simple Date Optional MM-DD-YYYY — for example, 08-10-2015. The date when the action window closes.
`startTime`	Simple Time Optional HH:MM, where HH is 00 to 24 and MM is 00 to 60. For each day the actions are to occur, the time for the action window to open.
`endTime`	Simple Time Optional HH:MM, where HH is 00 to 24 and MM is 00 to 60. For each day the actions are to occur, the time for the action window to close.
`color`	integer Optional The Hex color code to set the color of the item's display in the schedule panel. You give the number with the Hex prefix, or without it — `0x6dc066` and `6dc066` are both valid values.
`group`	UUID Optional The name or UUID for the group that sets the scope for this action window.
`action`	String Optional The action to perform. Can be one of: - `moveVM` - `startVM` - `changeVM` - `resizeVMForEfficiency` - `resizeVMForPerformance`

`actionMode` String

Optional

The action mode. Can be one of:

- MANUAL
- AUTOMATED

`recur` String

Optional

A string that describes the recurrence for this action window. You can use a string as follows: The default value is `NO`. To specify recurrence, give a string that is a set of attribute/value pairs separated by semicolons.

- `NO` — Turns off recurrence
- `DAILY`
- `WEEKLY` — For weekly recurrence, you also specify the day of the week, as follows: `WEEKLY-MO`. The day tokens are `MO`, `TU`, `WE`, `TH`, `FR`, `SA`, and `SU`.
- `MONTHLY` — In this case, the day of the month is the day specified in the date you specified for the `startDate`.

For example, `recur=WEEKLY-SA`.

`note` String

Optional

A description string for this scheduled action window. The string must be plain ASCII.

EXAMPLE: Edit an existing action window to change the end date

URL

```
https://administrator:xxyx123@10.10.123.45/vmturbo/api/schedules/_6dX-
0N_DEeSzLoNDXeGKHg?endDate=10-10-2015
```

Returns

On success, returns the UUID of the edited action window. Note that this UUID will be different than the UUID you passed to identify the action window.

POST: schedules

Create a new scheduled action window.

```
vmturbo/api/schedules
--data name=$string&startDate=$SimpleTime&endDate=$SimpleTime&startTime=$SimpleTime&
endTime=$SimpleTime&color=$integer&group=$uuid&action=$actionType&actionMode=$modeType
&recur=$string&note=$string
```

Create a new scheduled action window.

NOTE: The start date and time must be earlier than the end date and time.

`name`	String Required The action window's name.
`startDate`	Simple Date Required MM-DD-YYYY — for example, 04-10-2015. The date when the action window opens.
`endDate`	Simple Date Required MM-DD-YYYY — for example, 08-10-2015. The date when the action window closes.
`startTime`	Integer Required Integer for the hour of the day, from 0 to 24. For each day the actions are to occur, the time for the action window to open.
`endTime`	Integer Required Integer for the hour of the day, from 0 to 24. For each day the actions are to occur, the time for the action window to close.
`color`	integer Optional The Hex color code to set the color of the item's display in the schedule panel. You give the number with the Hex prefix, or without it — `0x6dc066` and `6dc066` are both valid values.
`group`	UUID or String Optional The name or UUID for the group that sets the scope for this action window.
`action`	String Required The action to perform. Can be one of: - `moveVM` - `startVM` - `changeVM` - `resizeVMForEfficiency` - `resizeVMForPerformance`
`actionMode`	String Required The action mode. Can be one of: - MANUAL - AUTOMATED

`recur` String

Optional

A string that describes the recurrence for this action window. You can use a string as follows: The default value is `NO`. To specify recurrence, give a string that is a set of attribute/value pairs separated by semicolons.

- `NO` — Turns off recurrence. The default value — if you don't specify recurrence this will be a one-time action window.
- `DAILY`
- `WEEKLY` — For weekly recurrence, you also specify the day of the week, as follows: `WEEKLY-MO`. The day tokens are `MO`, `TU`, `WE`, `TH`, `FR`, `SA`, and `SU`.
- `MONTHLY` — In this case, the day of the month is the day specified in the date you specified for the `startDate`.

For example, `recur=WEEKLY-SA`.

`note` String

Optional

A description string for this scheduled action window. The string must be plain ASCII.

EXAMPLE: Create a new action window

URL

```
https://administrator:xxx14DY55@10.10.123.45/vmturbo/api/schedules?
name=MyNewSchedule&startTime=11:00&endTime=18:00&startDate=05-10-2015
&endDate=10-10-2015&action=moveVM&actionMode=MANUAL&recur=MONTHLY
&group=c848d60daf99c333059c9c7241cb1166861a87df
```

Returns

On success, returns the UUID of the new action window. Note that this UUID will be different than the UUID you passed to identify the action window.

DELETE: schedules

`vmturbo/api/schedules/{uuid}`

Delete a scheduled action window.

`{uuid}` String

Required

The action window's unique ID.

EXAMPLE: Delete a scheduled action window:

URL

```
https://usr:pwd@10.10.172.86/vmturbo/api/schedules/_Y7bfIIi2EeGrq73x6zQ
```

Returns

On success, returns the UUID of the deleted action window. Note that this UUID will be different than the UUID you passed to identify the action window.