Turbonomic 6.0
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

Getting Started with the Turbonomic REST API

Welcome to the Turbonomic REST API Programmer's Guide. This guide will help you to use the Turbonomic REST API as you script interactions with the Turbonomic software and develop integrations between Turbonomic and other software applications. This guide gives you details about using the REST resources that are exposed by the API.

This guide is not a comprehensive reference to the REST API. For a complete reference, see the Turbonomic REST SwaggerUI documentation. You can use this guide as a compliment to the SwaggerUI documentation.

To access the Turbonomic REST SwaggerUI, open a web browser to:

`http://<Your_Turbonomic _IP>/vmturbo/apidoc/`.

Authentication

To use the API, you must have a valid user account on the Turbonomic software. Also note that accounts can have different roles. The API will only execute commands that are valid for your user role. For example, to execute Turbonomic recommnded actions, your account must have a role of either `administrator` or `automator`.

The REST API uses basic authentication. For example, with a curl client you specify a `-u` argument with your credentials. For example:

```
curl -i -X GET -u "user:pwd" http://10.10.123.123/vmturbo/rest/users
```

To avoid passing credentials with every call, you can set a session cookie in your client. For example, when you call the API via curl, the response header includes a session cookie that is valid for 24 hours. See the `Set-Cookie` entry below.

```
$ curl -i -X GET -u "user:pwd" http://10.10.123.123/vmturbo/rest/users

HTTP/1.1 200 OK
Date: Fri, 05 May 2017 16:05:16 GMT
Server: Apache-Coyote/1.1
Cache-Control: no-cache, no-store, max-age=0, must-revalidate
Pragma: no-cache
Expires: 0
X-XSS-Protection: 1; mode=block
X-Frame-Options: DENY
X-Content-Type-Options: nosniff
Content-Type: application/json
Set-Cookie: JSESSIONID=C2685A10098DF998CA8C3B21305799E9; Expires=Sat, 05-May-2018
16:05:16 GMT; Path=/vmturbo/; HttpOnly
Vary: Accept-Encoding
Transfer-Encoding: chunked
```

Then you can set that cookie in the curl client and continue to use it for your session:

```
$ curl -X GET --cookie "JSESSIONID=C2685A10098DF998CA8C3B21305799E9" http://
10.10.123.123/vmturbo/rest/users
```

URI Structure

To use the Turbonomic REST API, your client will make HTTP requests to specific REST resources. The Turbonomic REST API supports the standard HTTP methods:

- GET
  Get lists of entities or data objects, get individual items.
- POST
  Create new objects in the Turbonomic environment, or specify filters for certain queries.
- PUT
  Incrementally modify existing etities or objects.
- DELETE
  Delete entities or objects.

The base URI structure for a Turbonomic REST API resource is:

`http://<Your_Turbonomic _IP>/vmturbo/rest/<resource_name>`

Some URIs take the unique identifier (UUID) of an entity to include that entity as part of the resource path.

Response Format

The Turbonomic REST API returns data as JSON objects. You can try out REST methods in the SwaggerUI documentation to see typical response bodies.

To access the Turbonomic REST SwaggerUI, open a web browser to:

`http://<Your_Turbonomic _IP>/vmturbo`.

Introducing the Turbonomic REST API

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 section describes markets plans, and reservations in detail, along with the methods you can call to use them.

Turbonomic 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:

```
GET: /rest/markets
```

This listing returns the main real-time market, plus any plan markets that are in memory at the time of the call. Note that you can pass group UUIDs to set a scope for the call. In that case, the call returns only markets for plans that include the passed groups in their scope.

The Real-Time Market

The real-time market performs analysis and workload management on your environment. You can use the real-time market to access entities and get current or historical data about them.

The real-time market displayName is `Market`. This market should always be in the `RUNNING` state. Under no circumstances should you use the API to stop this market.

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 from or add entities to the real-time market.

For the real-time market, you can safely execute POST, PUT, and DELETE calls to:

- Post a query filter to get filtered lists of actions, notifications, and stats for the real-time market
- Post to create placement policies in the real-time market
- Delete to remove placement policies from the real-time market
- Put to edit placement policies on the real-time market
- Post a scenario to the real-time market to run a plan

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.

Plan Markets

Turbonomic can have markets other than the real-time market in memory. These other markets represent plans.

A plan market begins as a snapshot of the real-time market. You POST a scenario to the real-time market, and Turbonomic does two things:

- It makes a copy of the real-time market, to create a plan market. This plan market is just like the real-time market, except for any changes that were declared in the scenario. For example, if the scenario declares a scope for the plan, then the plan market only contains entities that are within that scope.
- It runs economic cycles (buy/sell cycles) against this plan market until there are no more meaningful improvements to be made. At this point the plan run is completed.

Note that once Turbonomic creates the plan market, that market stays in memory until you delete it. Also note that you can later apply a scenario to the plan market. This is how you run a plan on a plan. The logic flow is the same, and Turbonomic creates yet another plan market to run the analysis.

Internal-Use Plans

Turbonomic regularly 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.

A plan market that is for internal use includes the substring `_BasePlan` in the displayName. You should not modify these plan markets for any reason.

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 `<PlanType><userName>_<arbitraryID>`.

For example, a valid plan name is `CLOUD_MIGRATION_cud_1493140514716`. To find out which user owns this plan, you can parse out the user's name and query the API for that user's account information.

When you get a plan market, the response includes information such as:

- `uuid`: The market's identifier.
- `state`: Whether the plan succeeded or not. The state can be one of:
  - `CREATED`
  - `READY_TO_START`
  - `RUNNING`
  - `COPYING`
  - `SUCCEEDED`
  - `STOPPING`
  - `STOPPED`
  - `USER_STOPPED`
  - `DELETING`

  As long as a plan market is running, Turbonomic is still 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 plan results. You can make a PUT call to the market to stop a plan that is running.
- `scenario`: An object to describe the scenario used to run this plan. This includes the scenario UUID, displayName, and if specified, the scenario type.
- `unplacedEntitis`: Whether the plan could not place any entities. Can be true or false.
- `runDate`: When the plan was last started.
- `runComleteDate`: When the plan last completed.
- `violations`: An array of objects that describe violations in the plan. This includes the UUID for the affected entity, plus the entity displayName and a description of the violation.

For a plan market, you can execute the same POST, PUT, and DELETE methods that you would execute on the real-time market. These include:

- Post a query filter to get filtered lists of actions, notifications, and stats for the real-time market
- Post to create placement policies in the real-time market
- Delete to remove placement policies from the real-time market
- Put to edit placement policies on the real-time market
- Post a scenario to the real-time market to run a plan

You can also safely execute the following methods on a plan market:

- DELETE a plan market
- PUT to save or stop a plan market

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

Running Plans

A plan is a simulation or what-if scenario that explores the results of possible changes to your environment. To run these a plan, Turbonomic creates a snapshot copy of your real-time market and applias changes to it. It then uses the Economic Scheduling Engine to perform analysis on that plan market.

Before working with plans in the API, you should be familiar with plans via the GUI. You should know what plans can accomplish, and how to specify the plan settings such as:

- Plan scope
- Changes to worklaod in the environment (adding or removing VMs)
- Changes to supply (adding, removing, or reconfiguring hardware)
- Changes to placement (migrating to the cloud or a different cluster)
- Changes to constraints

To create a plan, you first specify a plan scenario. This is an object that contains all the plan settings. The scenario object contains an array of changes, and each change declares a setting that you want to make.

After you create a scenario, you POST it to a market. In most cases you will post it to the real-time market. When you post the scenario to the market, Turbonomic does two things:

- It makes a copy of the real-time market, to create a plan market. This plan market is just like the real-time market, except for any changes that were declared in the scenario. For example, if the scenario declares a scope for the plan, then the plan market only contains entities that are within that scope.
- It runs economic cycles (buy/sell cycles) against this plan market until there are no more meaningful improvements to be made. At this point the plan run is completed.

Note that once Turbonomic creates the plan market, that market stays in memory until you delete it. Also note that you can later apply a scenario to the plan market. This is how you run a plan on a plan. The logic flow is the same, and Turbonomic creates yet another plan market to run analysis on.

Saving and Deleting Plan Markets

Turbonomic supports saving plans. Users can save a plan via the GUI and you can save a plan via the API. When you save a plan, Turbonomic serializes the plan results so users can load the results into the GUI at a later time. To save a plan via the API, execute the following method:

`PUT: /rest/markets/_v0m-oCnaEeePgeXuo0RRbw?operation=save`

To delete a plan via the API, just execute a DELETE method on the plan market.

Creating Scenarios

The first step for a plan is to create the scenario. Note that a scenario is entirely separate from the plan. You create a scenario and then POST it to a market to create the plan. But the scenario still exists in memory, and you can apply it to a different market if you wish. For example, you can apply the same scenario to the real-time market at different times.

The API includes the `Scenarios` resource that you can use to create and edit scenarios. To create a scenario, use the `POST: /rest/scenarios/{name}` method. This method takes three parameters:

- `scope`
  An array of group UUIDs. For a scope of multiple groups, you must specify groups of the same type.
- `projection_days`
  For a projection scenario, the days to mark each projection period
- `input`
  A ScenarioApiInputDTO that can specify all the settings for a scenario.

The `input` parameter, a ScenarioApiInputDTO, is an object that you can POST to the API to specify scenario settings. The API includes methods to create, edit, and delete scenarios. In most cases, you will create or edit a scenario by POSTing or PUTing a ScenarioApiInputDTO with changes to declare the scenario settings you want. Note that the API includes methods you can use to PUT many of these settings into a specific scenario by passing properties directly in the URL. However, you can also make these settings with the ScenarioApiInputDTO. Learning to use this DTO is the most consistent way to create and modify a scenario.

Note that as you create a scenario you can POST all of its settings in the ScenarioApiInputDTO, or you can create an incomplete scenario, and then PUT changes into it ater via different calls.

Creating a Projection Scenario

Projection plans calculate infrastructure requirements into the future, so your environment can accommodate changes to workload requirements over time. In a single plan you can specify the scope of the plan, how far into the future to project, and by what increments of time. For example, you can project the requirements of a specific data-center one year into the future, showing how requirements change at one-month intervals.

When you create a projection scenario via the API, you provide the projection periods as an array of the days from today that you want to project. For example, `0` for today, `30` for 30 days from today, `60` for 60 days from today, and so on.

To create a scenario for a projection plan:

- Scope the scenario
  A projection plan requires a scope to the scenario.
- Create a change in the plan of type `PROJECTION_PERIODS`
  This change identifes the plan periods that you want, as an array of days.

```
{
    "changes":[
    {
      "type”:"PROJECTION_PERIODS",
      "projectionDays":[0,30,60,90]
    }
  ]
}
```

- For any plan changes that you want to repeat on specific projection periods, specify a `projectionDays` array for that change. Note that the items in this array must match the `projectionDays` that you specified for the `PROJECTION_PERIODS` change.
  For example, assume you specified `"projectionDays":[0,30,60,90]` for the initial projection:
  - Valid: `"projectionDays":[0,30,60,90]`
  - Valid: `"projectionDays":[30,90]`
  - Not valid: `"projectionDays":[0,35,70,90]`

This listing shows a plan scenario that scopes the plan market, sets up a projection for four periods, and adds one VM to the inventory for two out of four projection periods:

```
{
   "displayName":"My Projection",
   "changes": [
     {
       "type": "SCOPE",
       "scope": [
         {
           "uuid": "5678a46e9716657be88b5e1217df91436e13e4ff",
           "className": "Group",
           "entitiesCount": 2,
           "groupType": "VirtualMachine",
         }
       ]
     },
     {
       "type": "PROJECTION_PERIODS",
       "projectionDays": [
         0,
         30,
         60,
         90
       ]
     },
     {
       "type": "ADDED",
       "targets": [
         {
           "uuid": "564d89f4-190b-de1b-cb28-3e2f2126ab48"
         }
       ],
       "value": "1",
       "projectionDays": [
         30,
         90
       ]
     }
   ]
 }
```

Working with Placement Policies

You can enable or disable placement policies in a plan. This is useful to see how the environment would change if you had a different set of constraints. It's important to note that a placement policy exists in the context of a market. To get a placement policy, you query the market that contains it, and to create a placement policy you create it for the given market.

If you want to enable or disable a placement policy in a plan, you specify that as a change in the scenario. Because the placement policy exists in a market, you have to know which market you will run the plan against, and get the policy from that market. For example, if you will run the plan against the real-time market, you would:

- Query the real-time market for the policy you want
- Give that policy's UUID in the scenario change object
- In the scenario change object, specify to enable or disable the poicy
- When the scenario is complete, POST the scenario to the real-time market

Remember that when you run a plan, Turbonomic creates a new plan market. This plan market will contain a copy of the placement policy, and will enable or disable it, depending on your setting.

What if you want to add a new placement policy to the plan? In that case you must create the placement policy in the market *before* you run the plan, get the policy's UUID, and then add it as a change to your scenario. When you create the policy in the market, you should disable it to avoid unintended changes to your market (and consequently to your environment). The plan will test how the market reacts to the enabled policy.

To create a policy, use `POST: /rest/markets/{uuid}/policies`. For more information, see Getting and Creating Placement Policies on page 33.

Calculating Reservations and Workload Placement

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.

To specify the type of workload you are planning to deploy, the reservation includes a workload template, and a deployment profile. The API includes separate resources for these objects.

A reservation object does not necessarily reserve resources for workloads. You can use the reservation object to:

- Calculate Deployment
  For the given status of the current environment, calculate the optimal placement for workloads. A typical use case is to get placement results and then pass them to an orchestration system that actually executes the deployments.
- Deploy Immediately
  Calculate the optimal placement for the workloads, and then execute the deployment immediately.
- Reserve Resources and Deploy
  Calculate the optimal placement for the workloads, reserve the resources for these workloads, and then deploy them at a later date.

Note that before using the API to work with reservations, you should understand how reservations work from the user interface.

With the API, you can perform the following:

- Get list of reservations
- See the current placement for the reservation workloads
- Create reservations
- Immediately deploy the reserved workloads
- Delete a current reservation

Getting Reservation Information

To get a list of current reservations, execute `/rest/reservations`. This returns a list of all active reservations. If you know the UUID of the reservation you want, you can pass it to get data for just that reservation.

Each reservation object fully describes it, including:

- Display name
- Status — can be:
  - `DEPLOYING` — Turbonomic is deploying the workload
  - `DEPLOY_SUCCEEDED` — The workload was successfully deployed
  - `IN_PROGRESS` — Placement calculation is in progress
  - `PLACEMENT_SUCCEEDED` — For a new reservation, the environment has sufficient resources to place the workload; if you specified a reserve date, this will be an active reservation
  - `PLACEMENT_FAILED` — For a newly created reservation, the environment doesn't have resources to place the workload; if you specified a reserve date, this will be an unfulfilled reservation
  - `RETRYING` — Turbonomic is trying to place the workload of an unfulfilled reservation
- Times the reservation was created, time to deploy, and time it will expire
- A description of the reserved workload
- The deployment profile that identifies the physical files that will be copied to deploy the workload, as well as optional placement limitations
- Statistics for the compute and storage resources the reservation sets aside

For example, this listing shows a reservation for one VM:


```
{
  "uuid": "_kWZHIDDmEeePgeXuo0RRbw",
  "displayName": "MyReservation",
  "count": 1,
  "status": "RESERVED",
  "reserveDateTime": "Thu May 04 16:27:29 UTC 2017",
  "expireDateTime": "Thu Aug 31 16:27:29 UTC 2017",
  "deployDateTime": "Thu Aug 31 16:27:29 UTC 2017",
  "reserveCount": 1,
  "demandEntities": [
    {
      "uuid": "_kWgb7TDmEeePgeXuo0RRbw",
      "displayName": "MyReservation_C0",
      "className": "VirtualMachine",
      "template": {
        "uuid": "T423f548d-cadc-e525-6df4-1f90724cf696",
        "displayName": "vsphere-dc3.eng.vmturbo.com::TMP-SUSE64",
        "className": "VirtualMachineProfile"
      },
```

```
"deploymentProfile": {
  "uuid": "_gHJ0ICXxEeePgeXuo0RRbw",
  "displayName": "DEP-SUSE64",
  "className": "ServiceCatalogItem"
},
"placements": {
  "computeResources": [
    {
      "stats": [
        {
          "name": "numOfCpu",
          "value": 1
        },
        {
          "name": "cpuSpeed",
          "value": 2603
        },
        {
          "name": "cpuConsumedFactor",
          "value": 0.5
        },
        {
          "name": "memorySize",
          "value": 2097152
        },
        {
          "name": "memoryConsumedFactor",
          "value": 0.75
        },
        {
          "name": "ioThroughput",
          "value": 0
        },
        {
          "name": "networkThroughput",
          "value": 0
        }
      ],
      "provider": {
        "uuid": "Virtual_ESX_42381da5-12fa-1e82-2f1c-887419380d43",
        "displayName": "hp-esx21.corp.vmturbo.com",
        "className": "PhysicalMachine"
      }
    }
  ],
  "storageResources": [
    {
      "stats": [
        {
          "name": "diskSize",
          "value": 18432.363
        },
        {
          "name": "diskIops",
          "value": 0
```

```
            }
          ],
          "provider": {
            "uuid": "10545c15-7687ef26",
            "displayName": "QS2:ESXDC3DS1",
            "className": "Storage"
          }
        }
      ]
    }
  }
 ]
}
```

Creating a Reservation

To create a reservation, you POST an input DTO that defines the reservation's:

- `action`: The action type — `PLACEMENT`, `RESERVATION`, or `DEPLOYMENT`
- `demandName`: The display name of the reservation — If you do not specify names for the added workloads, this will be the root name for new VMs
- `deploymantParameters`:
  - `deploymentProfileID`: The UUID of a deployment profile
  - `highAvailability`
  - `priority`
- `placementParameters`:
  - `constraintIDs`: An array of UUIDs for placement policies that will affect the calculated placement
  - `count`: The number of workloads to place
  - `entityNames`: An array of names for the placed VMs — The array length should equal `count`
  - `geographicRedundancy`: If `true` place the workloads on unique hosts, otherwise Turbonomic can place multiple workloads on the same host
  - `templateID`: The UUID of the template that you will use to place this workload — Note that the template must include a reference to the deployment profile that you specify in `deploymantParameters`
- `deployDateTime`: When to deploy the workloads — Only provide this for an action of type `DEPLOYMENT`
- `expireDateTime`: When to cancel a reservation if Turbonomic cannot place all the workloads by that date — for a `RESERVATION` action, Turbonomic automatically sets the deploy time to equal this time
- `reserveDateTime`: The time to calculate the workload placement and create the reservation — This time cannot be earlier than the time that you POST the reservation to the API

The following listing shows an input DTO that creates a reservation. It will place four workloads, and it gives specific names ot each one.

```
{
  "action": "RESERVATION",
  "demandName": "MyReservation",
  "expireDateTime": "2017-10-10T12:38:17+00:00",
  "parameters": [
    {
      "deploymentParameters": {
        "deploymentProfileID": "_c9CJMDDAEeePgeXuo0RRbw"
      },
      "placementParameters": {
        "geographicRedundancy": false,
        "count": 4,
        "entityNames": [
          "foo","bar","baz","bonk"
        ],
        "templateID": "_UKsnkJkSEeCHcOXEhzJExA"
      }
    }
  ],
  "reserveDateTime": "2017-05-04T18:22:12+00:00"
}
```

Async or Blocked Placement Calculation

When you POST a reservation, Turbonomic runs a plan to calculate the optimal placement of the workloads. Depending on the size of the reservation, this can take a significant amount of time. This POST method includes the `apiCallBlock` parameter that specifies whether to execute the call asynchronously or in a blocked mode. If you do not set this parameter, then the API assumes asynchronous by default:

- Blocked: `/rest/reservations?apiCallBlock=true`
- Async: `/rest/reservations?apiCallBlock=false`

When you execute in asynchronous mode, the response to your POST shows that the calculation is in progress:

```
{
    "uuid": "_0dPrYTDlEeePgeXuo0RRbw",
    "displayName": "MyReservation",
    "count": 4,
    "status": "IN_PROGRESS",
    "reserveDateTime": "Thu May 04 18:22:12 UTC 2017",
    "expireDateTime": "Tue Oct 10 12:38:17 UTC 2017"
}
```

To examine the placement reslts, GET the reservation using the UUID that the API gives in the response.

When you execute the call in blocked mode, the response body contains the full reservation, including the status to show whether placement succeeded or failed.

Using Deployment Profiles and Templates

To create a reservation, you must specify a template and a deployment profile. These specify the workload requirements, and identify where to get the physical files to deploy.

VM Templates specify the resources that will be available to the VM, including:

- VCPUs
- Virtual Memory
- Storage
- Network Throughput
- IOPS
- IO Throughput

Note that you must choose a template that is mapped to a Deployment Profile. To determine this, GET the template you're interested in and look for the `deploymentProfile` property in the template object. For example, assume `myTemplate.deploymentProfile` gives:

```
"deploymentProfile": {
  "uuid": "_f8mJ9yXxEeePgeXuo0RRbw",
  "displayName": "DEP-5ff7938ad18c33bdad6a8af6b42f347b",
  "className": "ServiceCatalogItem"
},
```

You can use the included UUID as a deployment profile for workloads based on this template.

Note that templates can be created by users, and Turbonomic also discovers templates that are created by the management services in your environment. For example, a hypervisor or a cloud service provider typically manages a number of its own templates — Turbonomic discovers these. You should never edit a discovered template. Also, if that template includes a deployment profile, then you can assume the deployment profile was discovered also.

To see whether a template is discovered, GET the template you're interested in and look for the `discovered` property. This will be `true` or `false`.

The Turbonomic REST API Resources

The REST API resources give you full access to the Turbonomic software. This is a complete API that exposes the full set of Turbonomic capabilities. In fact, the product user interface is implemented as a Turbonomic client that uses this API.

While the REST API resources are documented in the SwaggerUI, this section provides extra details for the more expressige resources in the API.

To access the Turbonomic REST SwaggerUI, open a web browser to `http://<Your_Turbonomic _IP>/vmturbo`.

actions

Actions are the cornerstone of Turbonomic. Before problems occur, Turbonomic identifies actions you can take to avoid those problems. By continually performing these actions, you can keep your virtual environment running within the desired state.

The `actions` resource gives you direct access to specific actions. You can inspect an action, see its related notifications (risks and opportunities), and accept or reject it.

From this resource you can get:

- A list of all actions that are currently pending in the real-time market
- A specific action object
- For a given action, full details of the associated risk that leads to the action

You can also choose to accept or reject a specific action.

When you accept an action, if the action's `actionMode` is `MANUAL`, then Turbonomic will execute the action. For a `RECOMMEND` action, accepting the action clears it from the list of pending actions. You should only accept such an action after you have executed it in your environment.

Note that you cannot accept an `AUTOMATIC` action (Turbonomic automatically accepts it) or a `DISABLED` action.

Note that the `actions` resource works with actions in the real-time market. It does not work with actions that are in any plan markets. To get actions from a specific market, access actions through that market's UUID. Similarly, to get actions for a scope, you can get actions through a group's UUID. To get actions associated with a specific entity you can get actions through the entity's UUID:

- `/rest/markets/{uuid}/actions`
- `/rest/markets/{uuid}/actions/stats`
- `/rest/markets/{uuid}/actions/{action_id}`
- `/rest/groups/{uuid}/actions`
- `/rest/groups/{uuid}/actions/stats`
- `/rest/groups/{uuid}/actions/{action_id}`
- `/rest/entities/{uuid}/actions`
- `/rest/entities/{uuid}/actions/stats`
- `/rest/entities/{uuid}/actions/{action_id}`

After you get a list of actions, you can further parse the returned data object by different properties such as `actionType` or `createTime`.

Getting Filtered Data

The API includes a number of resources that include a POST method to retrieve actions and action statistics. These methods return filtered results. To use these methods, you POST a DTO that specifies properties that you want to filter the data against. For example, you know you can get all the actions associated with a group. But by posting a filter DTO, you can refine the request to get:

- All actions that were created within a given time range
- Only actions of a certain type (move, start, resize, etc.)
- Actions that resolve a risk of a specific sub-category (for example, Efficiency Improvement), or severity (MAJOR, MINOR, etc.)

The Swagger documentation shows the JSON format for the given filter in the Model Schema for the inputDTO. For example, to get a list of PROVISION actions in a market, POST the following DTO to the `/rest/markets/{uuid}/actions` resource:

```
{
  "actionTypeList": [
    "PROVISION"
  ]
}
```

deploymentprofiles

The Deployment Profile specifies physical details about how to deploy VMs from a given template. Turbonomic uses deployment profiles with reservations in order to actually deploy new instances of a given VM template.

You can use the `deploymentprofiles` resource to:

- Get lists of deployment profiles
- Create new deployment profiles
- Edit or delete deployment profiles

NOTE: Turbonomic can discover deployment profiles that have been created on hypervisors or cloud services. Discovered profiles use the following naming convention for the display name: `DEP-<ArbitraryId>`. For example, `displayName="DEP-7ab9971cd93031d1b23133275f878e17"`. Note that a discovered profile should be mapped to at least one discovered template. You should never delete or modify these discovered profiles. In the user interface they are read-only.

entities

Each market manages a set of entities. Using a market's UUID, you can get a list of entities managed by that market. For each entitiy you can get a full range of data, including the resources it buys and sells, the providers it buys resources from, actions for the entity, and other associated information.

Remember that more than one market can be resident in memory at a given time — Turbonomic maintains a real-time market, and there can also be plan markets in memory. 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 never 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
- You *can* add or remove entities in a plan market

An interesting point to about entities and markets — A plan market can contain entities that are also managed by the real-time market. These entities have the same stats in both markets. Turbonomic independently performs analysis on these entities in both markets.

Getting Entities

To get a list of entities, you start with the object that contains those entities. This is usually a market or a group. You work with entities via their UUIDs — A list of entities gives you a list of UUIDs.

You can also get entities from other sources, including:

- Entity relationships
  For example, entities that provide resources to a consumer entity.
- Entities associated with actions and notifications
- Entities in reservations
- The entities presented in a supply chain object

Getting Entity Data

You can get general information about an entity via the `rest/entities/{uuid}` resource, and you can also get specific data from an entity. The API provides the following resources:

- `rest/entities/`
  A list of links to sources of entities lists.
- `rest/entities/{uuid}`
  Specific data for an entity. This includes a list of providers for that entity, and consumers that buy resources from the entity.
- `rest/entities/{uuid}/actions`
  Any actions that are associated with the entity. You can also get action stats via `rest/entities/{uuid}/actions/stats`.
- `rest/entities/{uuid}/groups`
  Any groups that this entity is a membert of.
- `rest/entities/{uuid}/notifications`
  Any notifications (risks) associated with the entity. You can also get notification stats via `rest/entities/{uuid}/notifications/stats`.
- `rest/entities/{uuid}/policies`
  Any workload placement policies that are set for this entity.
- `rest/entities/{uuid}/settings`
  The overall policy settings that are in effect for this entity. For example, you can set the action modes globally or for a scope of entities, and these settings show which action modes are in effect for the current entity.
- `rest/entities/{uuid}/stats`
  THe max, min, average, and total values for this entity's biught and sold resources.
- `rest/entities/{uuid}supplychains`
  The supply chain for the given entity. This shows the consumers and providers for this entity in a form that maps to the supply chain that displays in the user interface.

Getting Filtered Data

The API includes a number of resources that include a POST method to retrieve actions and action statistics. These methods return filtered results. To use these methods, you POST a DTO that specifies properties that you want to filter the data against. For example, you know you can get all the actions associated with a group. But by posting a filter DTO, you can refine the request to get:

- All actions that were created within a given time range
- Only actions of a certain type (move, start, resize, etc.)
- Actions that resolve a risk of a specific sub-category (for example, Efficiency Improvement), or severity (MAJOR, MINOR, etc.)

The Swagger documentation shows the JSON format for the given filter in the Model Schema for the inputDTO. For example, to get a list of PROVISION actions in a market, POST the following DTO to the `/rest/markets/{uuid}/actions` resource:

```
{
  "actionTypeList": [
    "PROVISION"
  ]
}
```

The filtered requests you can post for entities include:

- /rest/entities/{uuid}/actions
  The actions that associated with the entity. Filter the request by:
  - Action modes
  - Action state
  - Action type
  - Cleared (t/f)
  - Start/end time
  - Risk severity
  - Risk category
- /rest/entities/{uuid}/actions/stats
  The stats for actions associated with the entity. For different times in the history for the entity, the stats show the number of actions at that time, and the investment cost for the actions. You can set the time range for this data request, and filter which actions to count.

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 for display in charts or for processing in plans. In the Turbonomic user interface, you can use groups to set scope for:

- Views in the GUI
- Plans
- Deployment Profiles (to limit deployment and reservation recommendations)
- User accounts
- Placement Policies
- Action settings

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

- Users
- User groups
- Planning scenarios

In addition, you can use the groups resource to set a scope on calls to get entities and actions.

Group Types

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

```
GET: /rest/groups
```

The returned list includes different classes of groups, with each type identified by the item's `className`. The group classes 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 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`, `RefMetaGroup`, and `MetaGroup`
  Groups used internally by Turbonomic.

In addition, each group has a `groupType` property that identifies the type of members that group contains. Remember that when you create a group in the user interface or via the API, you cannot add more than one type of member. The different group types include:

- `Cluster`
  A group of clusters. For example, you could create a group of all PM clusters that include the name "Development".
- `ServiceEntity`
  An internal group type reserved for groups that Turbonomic discovers. Note that the group will still contain members of only one type.

- Entity class name
  For a group of entities, the entity type. For example:
  - `VirtualMachine`
  - `PhysicalMachine`
  - `DiskArray`
  - `StorageController`
  - `Switch`
  - `Storage`
  - `Application`
  - `VirtualDataCenter`
  - `DataCenter`
  - `Container`
  - `ContainerPod`
  - `Network`
  - `DPod`
  - `VPod`
  - `Chassis`
  - `IOModule`
  - `Internet`
  - `LogicalPool`
  - `LoadBalancer`

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 `className` of `Cluster` or `StorageCluster`.

Getting Groups

When you execute a GET method on `/rest/groups`, the API returns a list of all the groups that are defined in Turbonomic. You can parse through this data to get all the groups of a given type.

Getting Custom Groups

Turbonomic collects all custom groups in a special group with a display name of "My Groups". To get a list of custom groups:

1. **Get all groups.**
   Execute `GET: /rest/groups`. This returns a list of all the groups in Turbonomic.
2. **Find the group named "My Groups".**
3. **Execute** `GET: /rest/groups/{uuid}/members` **with the UUID of that group.**
   This returns all the custom groups in Turbonomic.

Navigating in Groups of Groups

With groups of groups, it's possible to have multiple levels of nested groups. The API provides methods to move up or down the hierarchy.

To get child groups, use `GET: /rest/groups/{uuid}/members`.

To get parent groups, use `GET: /rest/groups/{uuid}/groups`.

Getting Group Entities

Each group object includes an `entitiesCount` property to tell you how many entities are in the group. For a group of entities (VirtualMachine, PhysicalMachine, DiskArray, etc.), the entity count is equal to the number of members in the group. For a group of groups the members are individual groups, and the entity count is the total of entities in all the member groups. For this reason, you should know the type of group before you get its members.

To get all the entities in a group, execute `GET: /rest/groups/{uuid}/entities` with the UUID of the group.

Note that for a group of groups, this returns all the entities in all the member groups. There is no guarantee that the order of the returned entities will map to the member groups in any way. To get the entities from a group of groups, it's best to get a list of the member groups (via `GET: /rest/groups/{uuid}/members`), and then get the entities from each one.

Getting Data Scoped by Group

An important use of groups is to set a scope to the data you work with. The API provides resources that use a group UUID to scope the data they return. Use these resources to get scoped lists of:

- Actions
- Action stats
- Notifications
- Notification stats
- Policies
- Settings
- Supply chains

Getting Filtered Data

The API includes a number of resources that include a POST method to retrieve actions and action statistics. These methods return filtered results. To use these methods, you POST a DTO that specifies properties that you want to filter the data against. For example, you know you can get all the actions associated with a group. But by posting a filter DTO, you can refine the request to get:

- All actions that were created within a given time range
- Only actions of a certain type (move, start, resize, etc.)
- Actions that resolve a risk of a specific sub-category (for example, Efficiency Improvement), or severity (MAJOR, MINOR, etc.)

The Swagger documentation shows the JSON format for the given filter in the Model Schema for the inputDTO. For example, to get a list of PROVISION actions in a market, POST the following DTO to the `/rest/markets/{uuid}/actions` resource:

```
{
  "actionTypeList": [
    "PROVISION"
  ]
}
```

The filtered requests you can post for groups include:

- /rest/groups/{uuid}/actions
  The actions that associated with the entities in a group. Filter the request by:
  - Action modes
  - Action state
  - Action type
  - Cleared (t/f)
  - Start/end time
  - Risk severity
  - Risk category
- /rest/groups/{uuid}/actions/stats
  The stats for actions associated with the entities in a group. For different times in the history for the group, the stats show the number of actions at that time, and the investment cost for the actions. You can set the time range for this data request, and filter which actions to count.

Creating Dynamic Groups

In the user interface, you create dynamic groups by choosing the entity type, specifying match criteria, and assigning a group name. With the API you POST the same information to the `/rest/groups` resource. For example:

```
{
  "displayName":"AllVMs",
  "groupType": "VirtualMachine",
  "isStatic": false,
  "criteriaList": [
      {
        "expVal": ".*",
        "expType": "EQ",
        "caseSensitive": false,
        "filterType": "vmsByName"
      }
  ]
}
```

This DTO specifies a dynamic group of virtual machines that matches every VM in the environment. For a dynamic group, the DTO must specify `"isStatic": false`.

The group DTO uses an array of match criteria. In this way, you can set up multiple search filters. Note that each criterion must apply to the same type of entity. To set up a search criterion, you provide:

- The match expression as `expVal`
- The match type (EQ, NEQ, GT, LT, GTE, LTE) as `expType`
- For strings, whether the match is case sensitive as `caseSensitive`
- The type of match to make as `filterType`

The `filterType` property determines how to match the group members. Note that the filter types you can use are different, depending on the type of entities you are adding to the group, as follows:

Group Type:	Filter Types:
VirtualMachine	• vmsByName • `vmsByPMName` VMs with matching display names • `vmsByStorage` VMs connected to storage with a matching name • `vmsByNetwork` VMs that are within a matching network • `vmsByApplication` VMs that host a matching application • `vmsByDC` VMs that are members of datacenters with matching display names • `vmsByVDC` VMs that are members of virtual datacenters with matching display names • `vmsByDCnested` VMs that are members of matching virtual datacenters within a hierarchy • `vmsByNumCPUs` VMs with the matching number of VCPUs • `vmsByMem` VMs with matching memory allocation • `vmsByGuestName` VMs with a Guest OS that has a matching name • `vmsByAltName` • `vmsByClusterName` VMs hosted on PMs that are members of the given cluster • `vmsByDiskArrayName` VMs connected to storage hosted on the given disk array • `vmsByTag` VMs with a matching tag and value
DataCenter	• `datacentersByName` Datacenters with a matching display name • `datacentersByTag` Datacenters with a matching tag and value
Cluster	• `clustersByTag` Clusters with a matching tag and value • `clustersByName` Clusters with a matching display name

Group Type:	Filter Types:
VirtualDataCenter	• `vdcsByName` Virtual datacenters with a matching display name • `vdcsByVDCName` Consumer virtual datacenters hosted by a matching provider VDC • `vdcsByTag` Virtual datacenters with a matching tag and value
PhysicalMachine	• `pmsByName` PMs with matching display names • `pmsByStorage` PMs connected to the named storage • `pmsByNetwork` PMs connected to the named network • `pmsByNumVms` PMs that host the number of VMs specified in the expression • `pmsByDC` PMs that are members of datacenters with matching display names • `pmsByMem` PMs with matching memory capacity • `pmsByNumCPUs` Add PMs with a matching count of CPUs • `pmsByVendorName` PMs with matching vendor names • `pmsByCPUModel` PMs with matching CPU model names • `pmsByModel` PMs with matching model names • `pmsByTimezone` PMs running in the given time zone • `pmsByClusterName` PMs that are members of the named cluster • `pmsByTag` PMs with a matching tag and value
Storage	• `storageByName` Storage with a matching display name • `storageByVMs` Storage that provides resources to the matching VMs • `storageByDC` Storage that is hosted by the matching datacenter • `storageByPMCluster` Storage that provides resources to a matching host cluster • `storageByTag` Storage with a matching tag and value • `storageClustersByName` Storage clusters with a matching display name

Group Type:	Filter Types:
Application	• `appsByName` Applications with a matching display name • `appSrvsByName` Application servers with a matching display name • `databaseByName` Database servers with a matching display name • `vappsByName` Virtual applications with a matching display name
Group	• `groupsByName` Groups with a matching display name
DiskArray	• `diskarrayByName` Disk arrays with a matching display name
StorageController	• `storagecontrollerByName` Storage controllers with a matching display name
Switch	• `switchByName` Switches with a matching display name
Container	• `containersByName` Containers with a matching display name • `containerpodsByName` Container pods with a matching display name • `containersByVMName` Containers hosted by VMs with a matching name • `containerpodsByVMName` Container pods hosted by VMs with a matching name

Creating Static Groups

In the user interface, you create static groups by choosing the entity type, then specifying a list of entities to add to the group. With the API you POST the same information to the `/rest/groups` resource. For example:

```
{
  "displayName":"Two_StaticVMs",
  "groupType": "VirtualMachine",
  "isStatic": true,
  "memberUuidList": [
    "423fde6b-8491-6e56-f43d-33aeb288efc1",
    "423fa4a2-61d6-05ee-a4f8-15d995a4fcf4"
  ]
}
```

This DTO specifies a static group of virtual machines that are specified in the `"memberUuidList` array. For a static group, the DTO must specify `"isStatic": true`.

markets

The `markets` resource gives you access to all the entities that Turbonomic manages. In Turbonomic, a market is a management domain. At any one time, there can be more than one market resident in memory:

- One Real-time Market
  The real-time market performs analysis and workload management on your environment. You can use the real-time market to access entities and get current or historical data about them.
  The real-time market displayName is `Market`. This market should always be in the `RUNNING` state. Under no circumstances should you use the API to stop this market.
- Multiple Plan Markets
  Plan markets are snapshots that are ultimately based on the real-time market. Individual users create plans to run what-if scenarios on the environment. You can also create plans via the API. A plan market stays resident in memory until a user delets it. Note that you can save a plan, and Turbonomic can choose to remove a saved plan market from memory.
  While you can base a plan on the real-time market, that plan usually includes changes such as setting scope, changing constraints, adding or removing workload or providers, etc. You can use the same markets methods to get and set information, but the plan topology and stats should be different from the real-time market.
  Plan markets can be in different states. It will be in the `RUNNING` state until the plan can find no improvements to that market, and then it stops. You stop a running plan via the API. In that case, the plan results will be incomplete.

Getting Markets

To get a list of all the markets in Turbonomic, execute:

```
GET: /rest/markets
```

This returns a list of all the markets, including the real-time market. Each market listing also includes information such as displayName, state (should always be `RUNNING` for the real-time market), unplaced entities, etc.

To get a listing of any plans that affect a specific group of entities, pass a `scope` parameter to this call. For the scope value, pass an array of group UUIDs:

```
GET: /rest/markets?scope=_GFRGcYh9Ed-Sl7QXyf-Qdw,_GXTGaYh3Wd-Sl9IXtw-Rvd
```

To get data for a single market, pass that market's UUID. Note that the returned data is the same as the data returned in the full list of markets:

```
GET: /rest/markets/1563754567248
```

Saving, Stopping, and Deleting Markets

NOTE: You should never save, stop, or delete the real-time market. The only changes you should make to the real-time market are POST and PUT placement policies, and POST a planning scenario to the real-time market.

It is safe to make changes directly to a plan market. These changes are:

- Save
  Direct Turbonomic to serialize the plan market and save it to disk so a user can load the plan results at a later date.

```
PUT: /rest/markets/1563754567248?operation=save
```

- Stop
  If the plan market is running, this aborts the planning process. The market retains any changes that were calculated up to the time that you stop the plan.

```
PUT: /rest/markets/1563754567248?operation=stop
```

- Delete
  A plan market remains in memory until you delete it. Even after you save a plan market, it can remain in memory unless Turbonomic deletes it to free resources.

```
DELETE: /rest/markets/1563754567248
```

Working with Plan Markets

A plan is a special instance of a market. To create a plan, you first specify a plan scenario. This is an object that contains all the plan settings. The scenario object contains an array of changes, and each change declares a setting that you want to make.

After you create a scenario, you POST it to a market. In most cases you will post it to the real-time market. When you post the scenario to the market, Turbonomic does two things:

- It makes a copy of the real-time market, to create a plan market. This plan market is just like the real-time market, except for any changes that were declared in the scenario. For example, if the scenario declares a scope for the plan, then the plan market only contains entities that are within that scope.
- It runs economic cycles (buy/sell cycles) against this plan market until there are no more meaningful improvements to be made. At this point the plan run is completed.

For more information about creating plans, see Running Plans on page 10.

Getting and Creating Placement Policies

A placement policy exists in the context of a market. Even if a plan market is an exact copy of the real-time market, the plan market has its own placement policies, each with its own UUID.

Before you work with placement policies via the API, you should understand how the work in the user interface. You should understand the types of placement policies you can create, and the effect of each on market analysis.

The API supports the following types of policies:

- `AT_MOST_N`: Only the given number of consumers can run on a single member of the providers group. This is set in the `capacity` property of the policy object.
- `BIND_TO_GROUP`: The consumers can only run on members of the provider group.

- `BIND_TO_COMPLEMENTARY_GROUP`: The consumers cannot run on any members of the provider group.
- `MUST_RUN_TOGETHER`: These consumers must run on the same provider entity.
- `AT_MOST_N_BOUND`: Only the given number of consumers can run on a single member of the providers group, AND The consumers can only run on members of the provider group.
- `MERGE`: Remove cluster boundaries for the specified clusters.
- `BIND_TO_GROUP_AND_LICENSE`: Create a license group.
- `BIND_TO_GROUP_AND_GEO_REDUNDANCY`: Specify how aggressively to keep consumers running on different members of the provider group.

When you get a policy, the returned object describes the policy type, as well as the consumer and provider groups. When you create a policy, you do not have to provide the full data. You provide:

- `buyerUuid`: The group of consumers for this policy.
- `sellerUuid`: The provider group for this policy.
- `type`: The policy type.
- `policyName`: A display name for this policy.
- `enabled`: Whether to enable the policy inthe market. Can be `true` or `false`.
- `capacity`: For an `AT_MOST_N` or `AT_MOST_N_BOUND` policy, the number of consumers to allow on a provider entity.
- `mergeType`: For a `MERGE` policy, the type of clusters to merge. Can be one of `Cluster`, `StorageCluster`, or `DataCenter`.
- `mergeUuids`: The groups that you want to merge. The group type must match the mergeType.

For example, to create a DON'T PLACE policy, post the following inputDto to the market:

```
{
  "buyerUuid": "f82dbbc2b3366052f3bc1ac8a68c9c06b0eb182a",
  "enabled": false,
  "policyName": "PolicyFromApi",
  "sellerUuid": "4a2f5f132ae690af147ccfd6ea9839e79da3db79",
  "type": "BIND_TO_COMPLEMENTARY_GROUP"
}
```

To edit a placement policy, PUT an inputDto to the given policy. Specify the changes you want in the inputDto.

scenarios

A scenario assembles configuration settings that you can use to set up and run a plan. To run a plan, you will apply the scenario to a market. Turbonomic will then run a plan based on the combination of the scenario and the market.

The settings you make in a scenario correspond to the plan settings you can make in the user interface. These include:

- Plan scope
- Changes to workload (adding, removing, or replacing VMs or containers)
- Changes to supply (adding, removing, replacing PMs or storage)
- Enable/disable lacement policies and other constraints
- Changes to action modes
- Enable/disable profisioning of supply
- Enable/disable resizing of workloads

The Scenario Object

When you get a scenario, the API returns an object that includes:

- `uuid`
  The unique identifier for this scenario.
- `displayName`
  By default, Turbonomic creates a display name that matches the scenario type. However, you can provide your own name when you create a scenario.
- `owners`
  An array of user accounts that can use the scenario. Currently, this is always the user that created the scenario.
- `type`
  The type of scenario. When a user creates a scenario in the GUI, this matches the given type that user chooses from the Plan Wizard.
- `changes`
  An array of the scenario settings that make changes to the market before running the plan. For example, changes that add workload, disable constraints, or set a different baseline.

Creating a Scenario

To create a scenario, use the `POST: /rest/scenarios/{name}` method. This method takes three parameters:

- `scope`
  An array of group UUIDs. For a scope of multiple groups, you must specify groups of the same type.
- `projection_days`
  For a projection scenario, the days to mark each projection period
- `input`
  A ScenarioApiInputDTO that can specify all the settings for a scenario.

Note that as you create a scenario you can POST all of its settings in the ScenarioApiInputDTO, or you can create an incomplete scenario, and then PUT changes into it ater via different calls.

The Changes Array in ScenarioApiInputDTO

The majority of settings you can make in a scenario are via an array of change objects in the ScenarioApiInputDTO, as follows:

```
{
  "changes":  [
    { ... }
  ]
}
```

Each change object in the array specifies a scenario setting. For each change, you specify the chane type, and then specify the actual settings. Each change type supports different settings, as you can see in the following examples:

- ADDED and REMOVED for adding and removing entities in the plan
  Provide the UUID of the entity you want to add or remove. This entity should be valid in the scope of the plan.

```
{
  "changes":[
    {
      "type”:"ADDED",
      "targets":[
        {
          "uuid":"xxx_EntityUUID"
        }
      ]
    }
  ]
}
```

- REPLACED for replacing entities with templates
  Provide the UUID of the entity you want to replace, and the UUID of the template you will replace it with.

```
{
  "changes":[
    {
      "type”:"REPLACED",
      "targets":[
        {
          "uuid":"xxx_EntityUUID"
        },
        {
          "uuid":"xxx_TemplateUUID"
        }
      ]
    }
  ]
}
```

- CONSTRAINTCHANGED to change constraints on VM placement. Choose the VMs this setting will affect.
  To specify a constraint change, set the following parameters:
  - `type`: The type of scenario setting. In this case, `CONSTRAINTCHANGED`.
  - `name`: The kind of constraint to set. Can be one of:
    * Empty string or undefined to specify all commodities
    * `ClusterCommodity`
    * `NetworkCommodity`
    * `DatastoreCommodity`
    * `StorageClusterCommodity`
    * `DataCenterCommodity`
  - `value`: Whether to enable or disable the constraints. Can be one of `true` or `false`.
  - `targets`: An array of VM entities or VM groups this constraint change will affect.

```
{
  "changes":[
    {
      "type”:"CONSTRAINTCHANGED",
      "value":"false",
      "name":"ClusterCommodity",
```

```
        "targets":[
          {
            "uuid":"xxx_VmGroupUUID"
          }
        ]
      }
    ]
}
```

- ADD_HIST to add VMs based on inventory changes in the last month. Specify one of `true` or `false`.

```
{
  "changes":[
     {
      "type”:"ADD_HIST",
      "enable":"true"
      }
  ]
}
```

- INCLUDE_RESERVED to include reserved VMs in the plan. Specify one of `true` or `false`.

```
{
  "changes":[
     {
      "type”:"INCLUDE_RESERVED",
      "enable":"true"
      }
  ]
}
```

- PROJECTION_PERIODS for a projection plan, to specify in days when to run each projection. This change takes an array of integers.

  Note that you can specify the `projectionDays` parameter for any other change, and that indicates that the change should take effect on those projection periods. Keep these two points in mind:

  - To create a projection scenario, you must include a PROJECTION_PERIODS change
  - For any other use of `projectionDays` in the same scenario, you must provide numbers that are members of the projection days you specify in this change

```
{
  "changes":[
    {
    "type”:"PROJECTION_PERIODS",
    "projectionDays":[0,30,60,90]
    }
  ]
}
```

- SET to make various settings in the scenario

  Settings you can make include:

  - `center`: The center of the desired state.
  - `diameter`: Distance from the center, as a diameter.
  - `description`: A description of the change. If you do not provide a description, Turbonomic generates one for you.
  - `index`: An index, starting from one, into the array of changes. You should not provide this when posting the DTO. But you can use this index to access changes in an existing scenario.

```
{
  "changes":[
    {
      "type”:"SET",
      "center":"80"
      "diameter":"20"
    }
  ]
}
```

- `SET_HIST_BASELINE` to set the market's used and peak utilization to values that were current at the given date

  By default, a scenario takes the baseline from the current market. Also, the default for this setting is to use the inventory that was current at the date you set. Specify the date in miliseconds

```
{
  "changes":[
    {
      "type”:"SET_HIST_BASELINE",
      "value":"1493251200000"
    }
  ]
}
```

- `SET_PEAK_BASELINE` to choose a peak baseline for the market resource utilization

  For each cluster in the plan’s scope, you can choose from the peak loads the scope has experienced in the past. In this way, you can set up a plan to run against the true peak workloads for each cluster. The peak baseline is just a snapshot of utilization at different times in the past. For each cluster in this baseline, the plan uses historical data to recreate the workload that cluster experienced at that time.

  For `value`, provide the time of the peak baseline that you want in miliseconds. For `targets`, specify the VM clusters that will have this baseline.

```
{
  "changes":[
    {
      "type”:"SET_PEAK_BASELINE",
      "value":"1493251200000",
      "targets":[
        {
          "uuid":"xxx_EntityOrGroupUUID"
        }
       ]
     }
  ]
}
```

- SET_USED to change the percentage of resources that an entity or group will use
  Provide the UUID of the entity or group that will have this used value, and give a positive or negative value from 0 to -+100.

```
{
  "changes":[
    {
      "type”:"SET_USED",
      "value": "-5",
      "targets":[
        {
          "uuid":"xxx_EntityOrGroupUUID"
        }
       ]
     }
   ]
}
```

- SET_UTILIZATION for host and storage groups or entities, the percentage of capacity that consumers can utilize
  Provide the UUID of the entity or group that will have this utilization value, and give a positive value from 0 to 100.

```
{
  "changes":[
    {
      "type”:"SET_UTILIZATION",
      "value": "75",
      "targets":[
        {
          "uuid":"xxx_HardwareEntityOrGroupUUID"
        }
      ]
    }
  ]
}
```

- SET_MAX_UTILIZATION to set the max percentage of a commodity's capacity that a VM can consume.
  Provide the UUID of the VM or group that will have this utilization value, and give a positive value from 0 to 100.

```
{
  "changes":[
    {
      "type”:"SET_MAX_UTILIZATION",
      "maxUtilType":"VMEM"
      "value": "75",
      "targets":[
        {
          "uuid":"xxx_VMEntityOrGroupUUID"
        }
      ]
    }
  ]
}
```

- SET_ACTION_SETTING to enable/disable provision or suspend for hosts and storage, and enable/disable resize for VMs

  Settings you can make include:

  - name:

    The type of action to enable or disable. Can be one of provision, suspend, or resize. Note that provision and suspend affect hosts and datastores, and resize affects VMs.
  - value:

    The entities you will apply this change to. You can give the classname for the type of entity you want to affect (VirtualMachine, PhysicalMachine, or Storage), or you can give the UUID of a group of entities. If you give the classname, then this setting affects all the entities of that type that are in the plan's scope.
  - enable:

    Whether to enable or disable this action. Can be one of true or false.

```
{
  "changes":[
    {
      "type”:"SET_ACTION_SETTING",
      "name": "provision",
      "value":"PhysicalMachine",
      "enable":"false"
    }
   ]
}
```

- SET_WORKLOAD_PLACEMENT to add placement policies to the plan or remove them from the plan

  Placement policies exist as members of a market. When you run a scenario on a market, Turbonomic creates a snapshot of the original market and runs the scenario on that plan market. This plan market also includes copies of the placement policies in the original market.

  Note that when you run the scenario against a market, then the resulting placement policies in the plan market will have new UUIDs.

  In the plan scenario you can enable or disable the placement policies that Turbonomic copies into the plan market. To do this, get the UUID of the policy from the market you will run the scenario against. Then add that policy as a target in the SET_WORKLOAD_PLACEMENT change for the scenario. As part of the change, specify whether to enable or disable the placement policy.

  In the user interface, you can also create a new policy and add it to the plan. To do this in the API, first create the policy in the market you will run the scenario against. Then add that policy ti the scenario and enable or disable it.

  To specify a placement policy change, set the following parameters:

  - value: The action to perform with the placement policy. Can be one of:
    * ADDED
    * ENABLED
    * DISABLED
    * REMOVED
  - targets: An array of UUIDs for the policies you want to perform the specified action on.

```
{
  "changes":
    [
      {
      "value":"DISABLED",
      "targets": [
        {
          "uuid": "xxxPolicyUUidFromTargetMarket"
```

```
          }
        ]
      }
    ]
  }
}
```

reservations

Turbonomic uses reservations to calculate the placement for these workloads you want to deploy, and manage reserved copies of those workloads so users can see their placement in the GUI. You can use the reservation object to:

- Calculate Deployment
  For the given status of the current environment, calculate the optimal placement for workloads. A typical use case is to get placement results and then pass them to an orchestration system that actually executes the deployments.
- Deploy Immediately
  Calculate the optimal placement for the workloads, and then execute the deployment immediately.
- Reserve Resources and Deploy
  Calculate the optimal placement for the workloads, reserve the resources for these workloads, and then deploy them at a later date.

You can GET a listing of all reservations, or you can GET a single reservation. The response body gives you the full description of the reservation, including:

- Display name
- Status — can be:
  - `DEPLOYING` — Turbonomic is deploying the workload
  - `DEPLOY_SUCCEEDED` — The workload was successfully deployed
  - `IN_PROGRESS` — Placement calculation is in progress
  - `PLACEMENT_SUCCEEDED` — For a new reservation, the environment has sufficient resources to place the workload; if you specified a reserve date, this will be an active reservation
  - `PLACEMENT_FAILED` — For a newly created reservation, the environment doesn't have resources to place the workload; if you specified a reserve date, this will be an unfulfilled reservation
  - `RETRYING` — Turbonomic is trying to place the workload of an unfulfilled reservation
- Times the reservation was created, time to deploy, and time it will expire
- A description of the reserved workload
- The deployment profile that identifies the physical files that will be copied to deploy the workload, as well as optional placement limitations
- Statistics for the compute and storage resources the reservation sets aside

Creating a Reservation

To create a reservation, you POST an input DTO that defines the reservation's:

- `action`: The action type — `PLACEMENT`, `RESERVATION`, or `DEPLOYMENT`
- `demandName`: The display name of the reservation — If you do not specify names for the added workloads, this will be the root name for new VMs
- `deploymantParameters`:
  - `deploymentProfileID`: The UUID of a deployment profile
  - `highAvailability`
  - `priority`
- `placementParameters`:
  - `constraintIDs`: An array of UUIDs for placement policies that will affect the calculated placement
  - `count`: The number of workloads to place
  - `entityNames`: An array of names for the placed VMs — The array length should equal `count`
  - `geographicRedundancy`: If `true` place the workloads on unique hosts, otherwise Turbonomic can place multiple workloads on the same host
  - `templateID`: The UUID of the template that you will use to place this workload — Note that the template must include a reference to the deployment profile that you specify in `deploymantParameters`
- `deployDateTime`: When to deploy the workloads — Only provide this for an action of type `DEPLOYMENT`
- `expireDateTime`: When to cancel a reservation if Turbonomic cannot place all the workloads by that date — for a `RESERVATION` action, Turbonomic automatically sets the deploy time to equal this time
- `reserveDateTime`: The time to calculate the workload placement and create the reservation — This time cannot be earlier than the time that you POST the reservation to the API

The following listing shows an input DTO that creates a reservation. It will place four workloads, and it gives specific names ot each one.

```
{
  "action": "RESERVATION",
  "demandName": "MyReservation",
  "expireDateTime": "2017-10-10T12:38:17+00:00",
  "parameters": [
    {
      "deploymentParameters": {
        "deploymentProfileID": "_c9CJMDDAEeePgeXuo0RRbw"
      },
      "placementParameters": {
        "geographicRedundancy": false,
        "count": 4,
        "entityNames": [
          "foo","bar","baz","bonk"
        ],
        "templateID": "_UKsnkJkSEeCHcOXEhzJExA"
      }
    }
  ],
  "reserveDateTime": "2017-05-04T18:22:12+00:00"
}
```

Async or Blocked Placement Calculation

When you POST a reservation, Turbonomic runs a plan to calculate the optimal placement of the workloads. Depending on the size of the reservation, this can take a significant amount of time. This POST method includes the `apiCallBlock` parameter that specifies whether to execute the call asynchronously or in a blocked mode. If you do not set this parameter, then the API assumes asynchronous by default:

- Blocked: `/rest/reservations?apiCallBlock=true`
- Async: `/rest/reservations?apiCallBlock=false`

When you execute in asynchronous mode, the response to your POST shows that the calculation is in progress:

```
{
    "uuid": "_0dPrYTDlEeePgeXuo0RRbw",
    "displayName": "MyReservation",
    "count": 4,
    "status": "IN_PROGRESS",
    "reserveDateTime": "Thu May 04 18:22:12 UTC 2017",
    "expireDateTime": "Tue Oct 10 12:38:17 UTC 2017"
}
```

To examine the placement reslts, GET the reservation using the UUID that the API gives in the response.

When you execute the call in blocked mode, the response body contains the full reservation, including the status to show whether placement succeeded or failed.

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 many types of targets, including:

- Hypervisors or other VM management services such as VMWare VCenter, or Citrix XenServer
- Private cloud managers such as CloudStack, OpenStack, or vCloud Director
- Public cloud managers such as AWS or Azure
- 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 complete information about targets, please be sure to review the Turbonomic Target Configuration Guide. You should not modify targets without a full understanding of how they work with Turbonomic.

Adding Targets

To discover entities in your environment, you add different targets to your Turbonomic installation. Turbonomic then uses these targets to collect data from your environment.

Turbonomic uses probes to connect to targets and discover their entities. The Turbonomic GUI provides forms for users to specify different target instances. Note that each probe can require different settings, and it's possible that different deployments of Turbonomic can support different sets of probes.

The REST API includes the `/rest/targets/specs` resource to generates a listing of all the targets that are valid for your installation. You can review this listing to see all the viable targets and the settings each one needs.

For example, this shows the listing for one of the targets returned by `/rest/targets/specs` -- a vCenter Hypervisor target:

```
[
  {
    "category": "Hypervisor",
    "type": "vCenter",
    "inputFields": [
      {
        "displayName": "Address",
        "name": "nameOrAddress",
        "isMandatory": true,
        "isSecret": false,
        "valueType": "STRING"
      },
      {
        "displayName": "Username",
        "name": "username",
        "isMandatory": true,
        "isSecret": false,
        "valueType": "STRING"
      },
      {
        "displayName": "Password",
        "name": "password",
        "isMandatory": true,
        "isSecret": true
      }
    ],
    "identifyingFields": [
      "nameOrAddress"
    ]
  },
  ...
]
```

From this listing, you can see the settings you need to make to connect to a vCenter Hypervisor target. In this case, you would provide an input DTO similar to the following, and pass it to the POST method for the `/rest/targets` resource:

```
{
  "category": "Hypervisor",
  "type": "vCenter",
  "inputFields": [
    {
      "name": "nameOrAddress",
      "value": "10.10.111.222"
    },
    {
      "name": "username",
      "value": "me.user"
    },
    {
      "name": "password",
      "value": "MyPassword"
    }
  ]
}
```

Target Validation and Discovery

When you add a target, Turbonomic validates the target (ensures a valid connection), and then performs an initial discovery of the entities that target manages. As long as the connection is valid, Turbonomic performs incremental discovery and executes actions through this target.

To check the status of a target, ecexute a GET method on the `/rest/targets/{uuid}` resource. (You can also get all targets, and find the target of interest in the returned data.) The data for a target includes a `status` property that shows `Validated` for a validated target, and a `lastValidated` property with the last successful validation in ISO86 format (YYY-MM-DD).

To execute a new validation and discovery pass, execute a POST method on the `/rest/targets/{uuid}` resource.

templates

Turbonomic uses templates to reserve resources and deploy workload in your environment, to calculate supply or demand changes in a plan, and to calculate workloads for cloud environments. You can use the `templates` resource to:

- Get lists of templates
- Create new templates
- Edit or delete templates

NOTE: Turbonomic can discover VM template data on hypervisors or cloud services. 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 these discovered templates. In the user interface they are read-only.

Creating Templates

You can create different types of templates, and for each type of template you provide different resource statistics. To create a template you pass a complete inputDTO to the POST method. To edit an existing template, you pass an inputDTO with the template's UUID, and with the parameters you want to change.

To illustrate how each template type uses different resources, here are some example inputDTOs for creating the different types of templates:

- VM Template

```
{
   "displayName": "MyVmTemplate",
   "className": "VirtualMachine",
   "vendor": "MyCo",
   "computeResources": [
     {
       "stats": [
         {
           "name": "numOfCpu",
           "value": 8
         },
         {
           "name": "cpuSpeed",
           "units": "MHz",
           "value": 3000
         },
         {
           "name": "cpuConsumedFactor",
           "units": "%",
           "value": 50
         },
         {
           "name": "memorySize",
           "units": "MB",
           "value": 15360
         },
         {
           "name": "memoryConsumedFactor",
           "units": "%",
           "value": 75
         }
       ]
     }
   ]
   "discovered": false
 }
```

- Host Template


```
{
   "displayName": "Cud_WonderHost2",
   "className": "PhysicalMachine",
   "price": 1000,
   "computeResources": [
     {
       "stats": [
         {
           "name": "numOfCores",
           "value": 2
         },
         {
           "name": "cpuSpeed",
           "units": "MHz",
           "value": 1024
         },
         {
           "name": "ioThroughputSize",
           "units": "MB/s",
           "value": 0
         },
         {
           "name": "memorySize",
           "units": "MB",
           "value": 2048
         },
         {
           "name": "networkThroughputSize",
           "units": "MB/s",
           "value": 0
         }
       ]
     }
   ],
   "infrastructureResources": [
     {
       "stats": [
         {
           "name": "powerSize",
           "value": 1
         },
         {
           "name": "spaceSize",
           "value": 1
         },
         {
           "name": "coolingSize",
           "value": 1
         }
       ]
     }
   ],
   "discovered": false
 }
```

- Storage Template


```
{
  "displayName": "MyStorageTemplate",
  "className": "Storage",
  "price": 100,
  "storageResources": [
    {
      "stats": [
        {
          "name": "diskIops",
          "value": 50
        },
        {
          "name": "diskSize",
          "units": "GB",
          "value": 5
        }
      ]
    }
  ],
  "price": 100,
  "discovered": false
}
```


- Container Template


```
{
  "displayName": "MyContainerTemplate",
  "className": "Container",
  "description": "Ubuntu 13.10 Docker Container Profile",
  "image": "ubuntu",
  "imageTag": "13.10A",
  "cmdWithArgs": "echo Test",
  "computeResources": [
    {
      "stats": [
        {
          "name": "numOfCpu",
          "value": 0
        },
        {
          "name": "cpuSpeed",
          "units": "MHz",
          "value": 1024
        },
        {
          "name": "cpuConsumedFactor",
          "units": "%",
          "value": 50
        },
        {
          "name": "memorySize",
          "units": "MB",
```

```
            "value": 2048
          },
          {
            "name": "memoryConsumedFactor",
            "units": "%",
            "value": 75
          },
          {
            "name": "ioThroughput",
            "units": "MB/s",
            "value": 1
          },
          {
            "name": "networkThroughput",
            "units": "MB/s",
            "value": 1
          }
        ]
      }
    ],
    "discovered": false
  }
```

VM Templates and Deployment Profiles

One use for VM templates is to set up the workloads to deploy in a reservation. In order to do this, the VM template must have a deployment profile mapped to it. To see whether it has a deployment profile, GET the template you're interested in and look for the `deploymentProfile` property in the template object. For example, assume `myTemplate.deploymentProfile` gives:

```
    "deploymentProfile": {
      "uuid": "_f8mJ9yXxEeePgeXuo0RRbw",
      "displayName": "DEP-5ff7938ad18c33bdad6a8af6b42f347b",
      "className": "ServiceCatalogItem"
    },
```

To map a deployment profile to a template, add the profile's UUID to the input DTO as a `deploymentProfileId` property.