Getting Started with Workloads(WC)

This section will explain how to get started with the ComputeNext workload API.

For this tutorial we will use a simple command line interface tool from ComputeNext called *runcws*.

runcws is a JavaScript program for Node.js.

(CWS = ComputeNext Web Services)

It is basically a simple layer over the REST API, and it keeps track of the various ID's for you to make things easier.

It makes one REST call to the ComputeNext REST API and returns the results.

You could also use any other HTTP or REST test tool such as *curl*

NOTE: This tutorial assumes you are familiar with the basic concepts of the ComputeNext API, including instances and workloads.

The workloadAPI consists of a bunch of standard CRUD methods (Create/Retrieve/Update/Delete) plus some other methods used to control the planning, activation and deactivation of resources in the workload.

Tutorial Notes

The examples in this tutorial were done with an early version of the runcws tool and used HTTP (not HTTPS).

It is **strongly recommended** that you use **HTTPS** for all interactions with the ComputeNext API because Basic authentication is used.

If HTTPS is not used, your credentials will be in clear text "on the wire".

Some examples in this tutorial were done at different times, so sometimes the timestamps etc. may not exactly match between examples.

To execute the workload plan you must first enter your payment information into the ComputeNext website. If you do not have payment information you will receive a "403 Forbidden" error.

The examples were run on a Windows platform, things might be slightly different on Linux.

Download and Install runcws

runcws is a Node.js script so first you will need to install Node.js from here: http://nodejs.org/download/

Any current stable build should work.

runcws is installed using NPM:

```
C:\>npm install runcws
npm http GET https://registry.npmjs.org/runcws
npm http 304 https://registry.npmjs.org/runcws
runcws@1.0.3 runcws

C:\>cd runcws

C:\runcws>dir
 Volume in drive C has no label.
 Volume Serial Number is A000-0000

 Directory of C:\runcws

01/06/2014  04:31 PM    <DIR>          .
01/06/2014  04:31 PM    <DIR>          ..
01/06/2014  04:31 PM               333 current.json
01/06/2014  04:31 PM    <DIR>          demo
01/06/2014  04:31 PM            18,365 LICENSE
01/06/2014  04:31 PM             1,184 package.json
01/06/2014  04:31 PM               168 README.md
01/06/2014  04:31 PM            10,991 runcws.js
01/06/2014  04:31 PM             6,726 runcws.json
01/06/2014  04:31 PM    <DIR>          workloads
               6 File(s)         37,767 bytes
               4 Dir(s)  583,675,293,696 bytes free
```

You will need to run *node runcws.js* from this directory.

The *current.json* file contains current settings for the runcws script and needs to be writable.

The *demo* directory contains sample JSON files for the instance API tutorial.

The *workloads* directory contains sample JSON workload files for the workload API tutorial.

Get a list of the possible commands/methods that runcws provides by entering at the command line:

```
>node runcws.js
SYNTAX: node runcws <command>
EXAMPLE: node runcws show
----- commands (resource) -----
metadata (retrieve resource metadata)
query (query resources)
region (retrieve region details)
restrictions (retrieve image restrictions)
capabilities (retrieve resource capabilities)
resource (retrieve resource details)
action (retrieve resource actions)
validate (validate resource)
----- commands (instance) -----
createi (create instance from resource)
getr (retrieve request)
listr (retrieve multiple requests)
createimage (create an image instance from a VM instance)
geti (retrieve instance)
getir (retrieve instance (with refresh))
listi (retrieve multiple instances)
listiwl (retrieve all instances in the workload)
listitx (retrieve all instances in the transaction)
updatei (update instance)
deletei (delete instance)
----- commands (workload) -----
createwl (create workload)
clonewl (clone workload)
getwl (retrieve workload)
listwl (retrieve multiple workloads)
updatewl (update workload)
deletewl (delete workload)
updateel (create or update a workload element)
deleteel (delete workload element)
activate (plan workload activation)
deactivate (plan workload deactivation)
execute (execute workload plan)
steps (retrieve transaction steps)
errors (retrieve transaction errors)
status (retrieve transaction status)
cancel (cancel transaction)
----- commands (misc) -----
setr (set current requestId)
seti (set current instanceId)
setwl (set current workloadId)
settx (set current transactionId)
show (show current settings)
```

Note the first set of commands are used to query resources. The query command itself has its own set of options shown as follows when one enters the command:

```
>node runcws.js query
query (query resources)
ERROR: missing parameter: resource type (image | instanceType | virtualMachine | volumeStorage |
softwareType | keyPair | securityGroup | loadBalancer | package)
```

Entering the following command will query the virtual machines producing the same output as using the http GET method:

```
>node runcws.js query virtualMachine
```

Set Up runcws

Before you start with runcws, you will need to obtain your API keys for your ComputeNext user account.

Instructions for doing this are here: Obtaining API Keys

As part of the runcws installation there is a JSON file named *current.json*

Open this with a text editor and update the *apikey* and *apisec* properties.

List (Retrieve Multiple) Workloads

First, we will try and list your workloads. If this succeeds, you will know that you are communicating OK with the ComputeNext REST API endpoint.

All input and output to/from the API is in JSON format.

If you have no workloads this method will return an empty JSON array:

```
>node runcws.js listwl
listwl (retrieve multiple workloads)
options: {
  "url": "http://cws.computenext.com/api/workload",
  "method": "get",
  "auth": {
    "user": "63a25f81-15dd-481e-a4d4-c5ed9a80f93a",
    "pass": "<hidden>"
  }
}
----- RESULT -----
[]
```

If you already have one or more workloads defined, your output may look something like this:

```
>node runcws.js listwl
listwl (retrieve multiple workloads)
options: {
  "url": "http://cws.computenext.com/api/workload",
  "method": "get",
  "auth": {
    "user": "63a25f81-15dd-481e-a4d4-c5ed9a80f93a",
    "pass": "<hidden>"
  }
}
----- RESULT -----
[
  {
    "workloadId": "34452cb1-7523-4157-b70b-09a326721aa6",
    "ownerId": "d00f16e6-7503-4838-a658-8b60594ff285",
    "uniqueName": "wl-26820482",
    "name": "WL-26820482",
    "description": "WL-26820482",
    "created": "2013-12-17T18:07:06.549Z",
    "updated": "2013-12-17T18:07:21.523Z",
    "workloadStatus": "none",
    "hasPlan": false,
    "hasExecute": false
  },
  {
    "workloadId": "235e4fdf-b39c-40a9-aa4c-177965414e51",
    "ownerId": "d00f16e6-7503-4838-a658-8b60594ff285",
    "uniqueName": "wl-d2940014",
    "name": "WL-D2940014",
    "description": "WL-D2940014",
    "created": "2013-12-17T18:07:28.667Z",
    "workloadStatus": "none",
    "hasPlan": false,
    "hasExecute": false
  }
]
```

In this case there are two workloads. In this call, only the “header” information for the workloads are returned, not the full workload JSON.

The *workloadStatus* property can have the values “none” or “in-progress”. “in-progress” means that some activity is in progress, such as planning, activation or deactivation (see later).

“hasPlan” means that this workload has a workload *plan*. See later.

“hasExecute” means that this workload has an *execute* section. See later.

Create Workload

Once we have established that we are communicating OK with the ComputeNext REST API endpoint, we can try to create our first workload.

There are some sample workloads in the runcws installation directory under “workloads”.

For a description of the workload schema see here: Workload Schema and here: Workload Element Schema

For a description of the parameters required for the various resource types see here: Resource Types, Actions and Parameters

To create our first workload:

```
>node runcws.js createwl workloads\hello_vm.json
createwl (create workload)
options: {
  "url": "http://cws.computenext.com/api/workload",
  "method": "post",
  "json": {
    "name": "Hello VM",
    "description": "Workload 'hello world' for one VM",
    "metadata": {
      "test": "this is metadata for the entire workload - can be anything",
      "test1": "another line of metadata"
    },
    "elements": [
      {
        "name": "VM 1",
        "uri": "vm/hpcloud/nova/standard.small",
        "parameters": {
          "imageUri": "image/hpcloud/nova/ami-00000075",
          "keyPair": "KP 1",
          "securityGroups": [
            "SG 1",
            "SG 2"
          ]
        },
        "metadata": {
          "description": "hello world - my first virtual machine"
        }
      }
    ]
  },
  "auth": {
    "user": "63a25f81-15dd-481e-a4d4-c5ed9a80f93a",
    "pass": "<hidden>"
  }
}
### setting current workloadId: b4ee62da-8dff-4a6b-b39f-54acf26a3a6d
----- RESULT -----
{
  "workloadId": "b4ee62da-8dff-4a6b-b39f-54acf26a3a6d",
  "ownerId": "d00f16e6-7503-4838-a658-8b60594ff285",
  "uniqueName": "hellovm",
  "name": "Hello VM",
  "description": "Workload 'hello world' for one VM",
  "created": "2013-12-19T01:23:27.354Z",
  "metadata": {
    "test": "this is metadata for the entire workload - can be anything",
    "test1": "another line of metadata"
  },
  "elements": [
    {
      "name": "VM 1",
      "uri": "vm/hpcloud/nova/standard.small",
      "parameters": {
        "imageUri": "image/hpcloud/nova/ami-00000075",
        "keyPair": "KP 1",
        "securityGroups": [
          "SG 1",
          "SG 2"
        ]
      },
      "metadata": {
        "description": "hello world - my first virtual machine"
      }
    }
  ]
}
```

The workload JSON is echoed back in the result from the API call so we can check that everything is correct.

Retrieve Workload

We can now retrieve the workload we just created:

```
>node runcws.js getwl
getwl (retrieve workload)
options: {
  "url": "http://cws.computenext.com/api/workload/b4ee62da-8dff-4a6b-b39f-54acf26a3a6d",
  "method": "get",
  "auth": {
    "user": "63a25f81-15dd-481e-a4d4-c5ed9a80f93a",
    "pass": "<hidden>"
  }
}
----- RESULT -----
{
  "workloadId": "b4ee62da-8dff-4a6b-b39f-54acf26a3a6d",
  "ownerId": "d00f16e6-7503-4838-a658-8b60594ff285",
  "uniqueName": "hellovm",
  "name": "Hello VM",
  "description": "Workload 'hello world' for one VM",
  "created": "2013-12-19T01:23:27.354Z",
  "metadata": {
    "test": "this is metadata for the entire workload - can be anything",
    "test1": "another line of metadata"
  },
  "elements": [
    {
      "name": "VM 1",
      "uri": "vm/hpcloud/nova/standard.small",
      "parameters": {
        "imageUri": "image/hpcloud/nova/ami-00000075",
        "keyPair": "KP 1",
        "securityGroups": [
          "SG 1",
          "SG 2"
        ]
      },
      "metadata": {
        "description": "hello world - my first virtual machine"
      }
    }
  ],
  "workloadStatus": "none"
}
```

Clone Workload

Cloning a workload allows us to create another workload that is exactly the same as the original workload except for its name - because the workload name must be unique.

Clone the workload:

```
>node runcws.js clonewl MyFirstClone
clonewl (clone workload)
options: {
  "url": "http://cws.computenext.com/api/workload/clone/b4ee62da-8dff-4a6b-b39f-54acf26a3a6d",
  "method": "post",
  "json": {
    "name": "MyFirstClone"
  },
  "auth": {
    "user": "63a25f81-15dd-481e-a4d4-c5ed9a80f93a",
    "pass": "<hidden>"
  }
}
### setting current workloadId: 9c3bfffc-0d12-4ebb-b897-83b8ee567065
----- RESULT -----
{
  "workloadId": "9c3bfffc-0d12-4ebb-b897-83b8ee567065",
  "ownerId": "d00f16e6-7503-4838-a658-8b60594ff285",
  "uniqueName": "myfirstclone",
  "name": "MyFirstClone",
  "description": "Workload 'hello world' for one VM",
  "created": "2013-12-19T01:38:31.713Z",
  "metadata": {
    "test": "this is metadata for the entire workload - can be anything",
    "test1": "another line of metadata"
  },
  "elements": [
    {
      "name": "VM 1",
      "uri": "vm/hpcloud/nova/standard.small",
      "parameters": {
        "imageUri": "image/hpcloud/nova/ami-00000075",
        "keyPair": "KP 1",
        "securityGroups": [
          "SG 1",
          "SG 2"
        ]
      },
      "metadata": {
        "description": "hello world - my first virtual machine"
      }
    }
  ]
}
```

If we now list the workloads, we will see both the original workload (name = Hello VM) and the new “cloned” workload (name = MyFirstClone)

Update Workload

To demonstrate updating a workload, we will overwrite our cloned workload (MyFirstClone) with a completely different workload JSON (Hello VS):

```
>node runcws.js updatewl workloads\hello_vs.json
updatewl (update workload)
options: {
  "url": "http://cws.computenext.com/api/workload/9c3bfffc-0d12-4ebb-b897-83b8ee567065",
  "method": "put",
  "json": {
    "name": "Hello VS",
    "description": "Workload 'hello world' for one VS",
    "metadata": {
      "test": "this is metadata for the entire workload - can be anything",
      "test1": "another line of metadata"
    },
    "elements": [
      {
        "name": "VS 1",
        "uri": "vs/hpcloud/nova/standard.10",
        "parameters": {
          "sizeInGBytes": 1
        },
        "metadata": {
          "description": "hello world - my first volume storage"
        }
      }
    ]
  },
  "auth": {
    "user": "63a25f81-15dd-481e-a4d4-c5ed9a80f93a",
    "pass": "<hidden>"
  }
}
----- RESULT -----
{
  "ownerId": "d00f16e6-7503-4838-a658-8b60594ff285",
  "uniqueName": "hellovs",
  "name": "Hello VS",
  "description": "Workload 'hello world' for one VS",
  "updated": "2013-12-19T01:42:26.517Z",
  "metadata": {
    "test": "this is metadata for the entire workload - can be anything",
    "test1": "another line of metadata"
  },
  "elements": [
    {
      "name": "VS 1",
      "uri": "vs/hpcloud/nova/standard.10",
      "parameters": {
        "sizeInGBytes": 1
      },
      "metadata": {
        "description": "hello world - my first volume storage"
      }
    }
  ]
}
```

Get the workload:

```
>node runcws.js getwl
getwl (retrieve workload)
options: {
  "url": "http://cws.computenext.com/api/workload/9c3bfffc-0d12-4ebb-b897-83b8ee567065",
  "method": "get",
  "auth": {
    "user": "63a25f81-15dd-481e-a4d4-c5ed9a80f93a",
    "pass": "<hidden>"
  }
}
----- RESULT -----
{
  "workloadId": "9c3bfffc-0d12-4ebb-b897-83b8ee567065",
  "ownerId": "d00f16e6-7503-4838-a658-8b60594ff285",
  "uniqueName": "hellovs",
  "name": "Hello VS",
  "description": "Workload 'hello world' for one VS",
  "created": "2013-12-19T01:38:31.713Z",
  "updated": "2013-12-19T01:42:26.517Z",
  "metadata": {
    "test": "this is metadata for the entire workload - can be anything",
    "test1": "another line of metadata"
  },
  "elements": [
    {
      "name": "VS 1",
      "uri": "vs/hpcloud/nova/standard.10",
      "parameters": {
        "sizeInGBytes": 1
      },
      "metadata": {
        "description": "hello world - my first volume storage"
      }
    }
  ],
  "workloadStatus": "none"
}
```

Note that the workloadId has not changed, so we updated the name, description, metadata and elements properties of the original cloned workload.

Update Workload Element

Now, let’s add one new workload element (VM 2) to this workload:

```
>node runcws.js updateel workloads\vm_element.json
updateel (create or update a workload element)
options: {
  "url": "http://cws.computenext.com/api/workload/9c3bfffc-0d12-4ebb-b897-83b8ee567065/element",
  "method": "put",
  "json": {
    "name": "VM 2",
    "uri": "vm/hpcloud/nova/standard.small",
    "parameters": {
      "imageUri": "image/hpcloud/nova/ami-00000075",
      "keyPair": "KP 1",
      "securityGroups": [
        "SG 1",
        "SG 2"
      ]
    },
    "metadata": {
      "description": "hello world - my SECOND virtual machine"
    }
  },
  "auth": {
    "user": "63a25f81-15dd-481e-a4d4-c5ed9a80f93a",
    "pass": "<hidden>"
  }
}
----- RESULT -----
{
  "name": "VM 2",
  "uri": "vm/hpcloud/nova/standard.small",
  "parameters": {
    "imageUri": "image/hpcloud/nova/ami-00000075",
    "keyPair": "KP 1",
    "securityGroups": [
      "SG 1",
      "SG 2"
    ]
  },
  "metadata": {
    "description": "hello world - my SECOND virtual machine"
  }
}
```

Get the workload:

```
>node runcws.js getwl
getwl (retrieve workload)
options: {
  "url": "http://cws.computenext.com/api/workload/9c3bfffc-0d12-4ebb-b897-83b8ee567065",
  "method": "get",
  "auth": {
    "user": "63a25f81-15dd-481e-a4d4-c5ed9a80f93a",
    "pass": "<hidden>"
  }
}
----- RESULT -----
{
  "workloadId": "9c3bfffc-0d12-4ebb-b897-83b8ee567065",
  "ownerId": "d00f16e6-7503-4838-a658-8b60594ff285",
  "uniqueName": "hellovs",
  "name": "Hello VS",
  "description": "Workload 'hello world' for one VS",
  "created": "2013-12-19T01:38:31.713Z",
  "updated": "2013-12-19T01:47:06.516Z",
  "metadata": {
    "test": "this is metadata for the entire workload - can be anything",
    "test1": "another line of metadata"
  },
  "elements": [
    {
      "name": "VM 2",
      "uri": "vm/hpcloud/nova/standard.small",
      "parameters": {
        "imageUri": "image/hpcloud/nova/ami-00000075",
        "keyPair": "KP 1",
        "securityGroups": [
          "SG 1",
          "SG 2"
        ]
      },
      "metadata": {
        "description": "hello world - my SECOND virtual machine"
      }
    },
    {
      "name": "VS 1",
      "uri": "vs/hpcloud/nova/standard.10",
      "parameters": {
        "sizeInGBytes": 1
      },
      "metadata": {
        "description": "hello world - my first volume storage"
      }
    }
  ],
  "workloadStatus": "none"
}
```

You can see that there was originally only the “VS 1” workload element, now the “VM 2” workload element has been added.

Delete Workload Element

To delete a workload element, the JSON that is sent must contain (at least) the workload element name.

Workload elements are identified by name only, and the workload element name must be unique in the workload.

Delete one element (VM 2):

```
>node runcws.js deleteel workloads\vm_element.json
deleteel (delete workload element)
options: {
  "url": "http://cws.computenext.com/api/workload/9c3bfffc-0d12-4ebb-b897-83b8ee567065/element",
  "method": "delete",
  "json": {
    "name": "VM 2",
    "uri": "vm/hpcloud/nova/standard.small",
    "parameters": {
      "imageUri": "image/hpcloud/nova/ami-00000075",
      "keyPair": "KP 1",
      "securityGroups": [
        "SG 1",
        "SG 2"
      ]
    },
    "metadata": {
      "description": "hello world - my SECOND virtual machine"
    }
  },
  "auth": {
    "user": "63a25f81-15dd-481e-a4d4-c5ed9a80f93a",
    "pass": "<hidden>"
  }
}
----- RESULT -----
{
  "name": "VM 2",
  "uri": "vm/hpcloud/nova/standard.small",
  "parameters": {
    "imageUri": "image/hpcloud/nova/ami-00000075",
    "keyPair": "KP 1",
    "securityGroups": [
      "SG 1",
      "SG 2"
    ]
  },
  "metadata": {
    "description": "hello world - my SECOND virtual machine"
  }
}
```

Get the workload:

```
>node runcws.js getwl
getwl (retrieve workload)
options: {
  "url": "http://cws.computenext.com/api/workload/9c3bfffc-0d12-4ebb-b897-83b8ee567065",
  "method": "get",
  "auth": {
    "user": "63a25f81-15dd-481e-a4d4-c5ed9a80f93a",
    "pass": "<hidden>"
  }
}
----- RESULT -----
{
  "workloadId": "9c3bfffc-0d12-4ebb-b897-83b8ee567065",
  "ownerId": "d00f16e6-7503-4838-a658-8b60594ff285",
  "uniqueName": "hellovs",
  "name": "Hello VS",
  "description": "Workload 'hello world' for one VS",
  "created": "2013-12-19T01:38:31.713Z",
  "updated": "2013-12-19T01:57:58.944Z",
  "metadata": {
    "test": "this is metadata for the entire workload - can be anything",
    "test1": "another line of metadata"
  },
  "elements": [
    {
      "name": "VS 1",
      "uri": "vs/hpcloud/nova/standard.10",
      "parameters": {
        "sizeInGBytes": 1
      },
      "metadata": {
        "description": "hello world - my first volume storage"
      }
    }
  ],
  "workloadStatus": "none"
}
```

You can see the “VM 2” workload element has now been removed from the workload.

Delete Workload

To delete the workload:

```
>node runcws.js deletewl
deletewl (delete workload)
options: {
  "url": "http://cws.computenext.com/api/workload/9c3bfffc-0d12-4ebb-b897-83b8ee567065",
  "method": "delete",
  "auth": {
    "user": "63a25f81-15dd-481e-a4d4-c5ed9a80f93a",
    "pass": "<hidden>"
  }
}
----- RESULT -----
{
  "ownerId": "d00f16e6-7503-4838-a658-8b60594ff285",
  "workloadId": "9c3bfffc-0d12-4ebb-b897-83b8ee567065",
  "deleted": true
}
```

The “deleted” property tells us whether the workload was actually deleted - or whether it was just not actually there:

```
>node runcws.js deletewl
deletewl (delete workload)
options: {
  "url": "http://cws.computenext.com/api/workload/9c3bfffc-0d12-4ebb-b897-83b8ee567065",
  "method": "delete",
  "auth": {
    "user": "63a25f81-15dd-481e-a4d4-c5ed9a80f93a",
    "pass": "<hidden>"
  }
}
----- RESULT -----
{
  "ownerId": "d00f16e6-7503-4838-a658-8b60594ff285",
  "workloadId": "9c3bfffc-0d12-4ebb-b897-83b8ee567065",
  "deleted": false
}
```

Plan Workload Activation

We have now deleted the workload that we cloned, so we need to set our workloadId used by runcws back to the original workload (Hello VM):

```
>node runcws.js setwl b4ee62da-8dff-4a6b-b39f-54acf26a3a6d
setwl (set current workloadId)
----- current settings -----
requestId: 190fe3a8-4d97-4de3-b6a9-c00ae7a4e1ec
instanceId: 3e9b4f1a-3e1e-4784-941a-feab27974b45
workloadId: b4ee62da-8dff-4a6b-b39f-54acf26a3a6d
transactionId: 8b3e4597-ab54-4b37-bcf0-3ebbbf1238f3
```

BTW, we can see the current settings used by runcws using the *show* command:

```
>node runcws.js show
show (show current settings)
----- current settings -----
requestId: 190fe3a8-4d97-4de3-b6a9-c00ae7a4e1ec
instanceId: 3e9b4f1a-3e1e-4784-941a-feab27974b45
workloadId: b4ee62da-8dff-4a6b-b39f-54acf26a3a6d
transactionId: 8b3e4597-ab54-4b37-bcf0-3ebbbf1238f3
```

We are now going to plan the activation of the workload.

In the workload API, activation (or deactivation) of a workload is a two phase process. First, the workload is planned, and then the plan is executed. This two phase process allows you to check what will be done before it is actually done, so you can verify that you are getting what you expected. If something fails, it also makes it easier to understand exactly what failed and why.

To plan activation:

```
>node runcws.js activate
activate (plan workload activation)
options: {
  "url":
"http://cws.computenext.com/api/workload/b4ee62da-8dff-4a6b-b39f-54acf26a3a6d/plan?action=activate",
  "method": "put",
  "auth": {
    "user": "63a25f81-15dd-481e-a4d4-c5ed9a80f93a",
    "pass": "<hidden>"
  }
}
----- RESULT -----
{
  "workloadStatus": "in-progress",
  "action": "plan-activate"
}
```

The workload is now in the process of planning. For a simple workload such as this it should be quite quick. For larger and more complex workloads this can take some time.

Get the workload:

```
>node runcws.js getwl
getwl (retrieve workload)
```

```
options: {
  "url": "http://cws.computenext.com/api/workload/b4ee62da-8dff-4a6b-b39f-54acf26a3a6d",
  "method": "get",
  "auth": {
    "user": "63a25f81-15dd-481e-a4d4-c5ed9a80f93a",
    "pass": "<hidden>"
  }
}
----- RESULT -----
{
  "workloadId": "b4ee62da-8dff-4a6b-b39f-54acf26a3a6d",
  "ownerId": "d00f16e6-7503-4838-a658-8b60594ff285",
  "uniqueName": "hellovm",
  "name": "Hello VM",
  "description": "Workload 'hello world' for one VM",
  "created": "2013-12-19T01:23:27.354Z",
  "metadata": {
    "test": "this is metadata for the entire workload - can be anything",
    "test1": "another line of metadata"
  },
  "elements": [
    {
      "name": "VM 1",
      "uri": "vm/hpcloud/nova/standard.small",
      "parameters": {
        "imageUri": "image/hpcloud/nova/ami-00000075",
        "keyPair": "KP 1",
        "securityGroups": [
          "SG 1",
          "SG 2"
        ]
      },
      "metadata": {
        "description": "hello world - my first virtual machine"
      }
    }
  ],
  "workloadStatus": "none",
  "plan": {
    "action": "activate",
    "ownerId": "d00f16e6-7503-4838-a658-8b60594ff285",
    "created": "2013-12-19T02:07:54.180Z",
    "expires": "2013-12-19T02:12:54.180Z",
    "elements": [
      {
        "name": "VM 1",
        "uri": "vm/hpcloud/nova/standard.small",
        "parameters": {
          "imageUri": "image/hpcloud/nova/ami-00000075",
          "keyPairId": "*0000_kp_create_kp1",
          "securityGroupIds": [
            "*0001_sg_create_sg1",
            "*0003_sg_create_sg2"
          ]
        },
        "metadata": {
          "description": "hello world - my first virtual machine",
          "name": "VM 1"
        },
        "resource": {
          "id": "vm_hpcloud_nova_standard-small",
          "uri": "vm/hpcloud/nova/standard.small",
          "resourceType": "vm",
          "provider": "hpcloud",
          "region": "nova",
          "providerResourceId": "Standard.small",
          "cpuSpeed": "1.2",
          "cpuCount": "2",
          "localStorage": "60",
          "ram": "2",
          "operatingSystemVersion": "64 Bit",
          "zone": "nova",
```

```
      "connectorType": "openStack.compute"
    }
  }
],
"serial": [
  {
    "parallel": [
      {
        "step": {
          "id": "0000_kp_create_kp1",
          "action": "kp.create",
          "uri": "kp/hpcloud/nova/standard",
          "metadata": {
            "name": "KP 1"
          },
          "sourceElement": "VM 1",
          "resource": {
            "provider": "hpcloud",
            "region": "nova",
            "resourceType": "kp"
          },
          "timing": {
            "min": 1.09,
            "avg": 1.82,
            "max": 2.11
          }
        }
      }
    ]
  },
  {
    "parallel": [
      {
        "step": {
          "id": "0001_sg_create_sg1",
          "action": "sg.create",
          "uri": "sg/hpcloud/nova/standard",
          "metadata": {
            "name": "SG 1"
          },
          "sourceElement": "VM 1",
          "resource": {
            "provider": "hpcloud",
            "region": "nova",
            "resourceType": "sg"
          },
          "timing": {
            "min": 1.03,
            "avg": 1.04,
            "max": 1.08
          }
        }
      },
      {
        "step": {
          "id": "0003_sg_create_sg2",
          "action": "sg.create",
          "uri": "sg/hpcloud/nova/standard",
          "metadata": {
            "name": "SG 2"
          },
          "sourceElement": "VM 1",
          "resource": {
            "provider": "hpcloud",
            "region": "nova",
            "resourceType": "sg"
          },
          "timing": {
            "min": 1.03,
            "avg": 1.04,
            "max": 1.08
          }
```

```
      }
    }
  ]
},
{
  "parallel": [
    {
      "step": {
        "id": "0002_sg_update_add_access_sg1",
        "action": "sg.update.add-access",
        "instanceId": "*0001_sg_create_sg1",
        "parameters": {
          "rules": [
            {
              "protocol": "tcp",
              "from-port": 22,
              "to-port": 22
            },
            {
              "protocol": "tcp",
              "from-port": 3389,
              "to-port": 3389
            },
            {
              "protocol": "icmp",
              "from-port": -1,
              "to-port": -1
            }
          ]
        },
        "sourceElement": "VM 1",
        "timing": {
          "min": 1.06,
          "avg": 1.4,
          "max": 3.1
        }
      }
    },
    {
      "step": {
        "id": "0004_sg_update_add_access_sg2",
        "action": "sg.update.add-access",
        "instanceId": "*0003_sg_create_sg2",
        "parameters": {
          "rules": [
            {
              "protocol": "tcp",
              "from-port": 22,
              "to-port": 22
            },
            {
              "protocol": "tcp",
              "from-port": 3389,
              "to-port": 3389
            },
            {
              "protocol": "icmp",
              "from-port": -1,
              "to-port": -1
            }
          ]
        },
        "sourceElement": "VM 1",
        "timing": {
          "min": 1.06,
          "avg": 1.4,
          "max": 3.1
        }
      }
    }
  ]
},
```

```
      {
        "parallel": [
          {
            "step": {
              "id": "0005_vm_create_vm1",
              "action": "vm.create",
              "uri": "vm/hpcloud/nova/standard.small",
              "parameters": {
                "imageUri": "image/hpcloud/nova/ami-00000075",
                "keyPairId": "*0000_kp_create_kp1",
                "securityGroupIds": [
                  "*0001_sg_create_sg1",
                  "*0003_sg_create_sg2"
                ]
              },
              "metadata": {
                "description": "hello world - my first virtual machine",
                "name": "VM 1"
              },
              "sourceElement": "VM 1",
              "timing": {
                "min": 90.34,
                "avg": 90.34,
                "max": 90.34
              }
            }
          }
        ]
      }
    ],
    "inventory": {
      "hpcloud": {
        "nova": {
          "quota": {
            "kp": {
              "count": "17.31"
            },
            "sg": {
              "count": "32.00"
            },
            "vm": {
              "count": "4.00",
              "cpuCount": "0.60",
              "localStorage": "1.80",
              "ram": "0.06"
            },
            "vs": {
              "count": "0.00",
              "sizeInGBytes": "0.00"
            }
          }
        }
      }
    },
    "summary": {
      "kp": 1,
      "sg": 2,
      "vm": 1
    }
  }
}
```


You can see that the “plan” section has now been added into the workload JSON and that it contains a lot of new properties.

- plan.action
- plan.expires
- plan.elements
- plan.serial
- plan.inventory
- plan.summary

plan.action

This indicates what kind of plan was requested - “activate” or “deactivate”.

plan.expires

The workload plan depends on the current state of the instances, so the plan has an expire time of 5 minutes. When the plan expires, it will be removed from the workload JSON.

plan.elements

This is a snapshot of the workload elements at the time they were used for the plan. At this point you can update the workload and add & delete elements to the *elements* section, so the *plan.elements*section preserves what was used for the plan. The *plan.elements* section within the plan is slightly different to the original *elements* section. It has been processed into a form that is useable by the instance API, because each step will be a call to the instance API. You may see references to other steps which start with an asterisk ‘*’. A *resource* property has been added to each element which contains a summary of the resource properties.

plan.serial

This is the start of the actual workload plan. The actual workload plan is a set of nested sections, some can be serial (executed one at a time) and some can be parallel (executed concurrently). It always starts with *plan.serial*. The serial/parallel sections contain workload steps, which are the steps required to provision the workload elements. See below for the properties of a workload step.

plan.inventory

The plan has a *plan.inventory* property which gives a percentage indication of how much of a resource quota will be used at this region once the workload is provisoned. For example, “inventory.hpcloud.nova.quota.kp.count” is 17.31, which means that 17.31 percent of the quota for key pairs will be reached for the hpcloud.nova region once this workload has been successfully provisioned.

Note that the quotas are not enforced. You may have a quota over 100 percent at some region and you will still be allowed to execute the workload plan - but it is likely that it will fail.

User limits are similar to quotas, but these are the limits on how many resource instances of a given type can be created by the user. These limits are enforced, and if you exceed these limits then there will be an *error* section added to the workload JSON with an explanation of the error.

plan.summary

This property summarizes how many resources of each resource type will be created when this plan is executed.

Workload Steps

A workload step can have similar properties to the workload element from which it was generated. However - one workload element can generate multiple workload steps.

- step.id - this consists of a 4 digit number plus some identifying information about the step - it uniquely identifies the step within the plan.
- step.action - this specifies the instance API action that will be performed for the step.
- step.sourceElement - this references the original workload element that caused this step to be generated.
- step.timing - this has the minimum, average and maximum time (in seconds) expected for this step.

How the Plan is Generated

The workload plan is generated as follows -

1. Key pairs (kp) are created. If you specify a kp by name in the workload element, the workload planner checks to see whether a kp instance with that name (in the instance metadata) already exists in the region. If it does, it uses it. If not, it adds a step to create the kp.
2. Security groups (sg) are created. If you specify an sg by name in the workload element, the workload planner checks to see whether that sg exists. If not, it adds a step to create the sg. It also adds a step to add SSH, RDP and “ping” ports to that sg.
3. Virtual machines (vm) are created.
4. Floating IP addresses (ip) are created - if the vm requires them. A step is added to create the ip, and then another step is added to associate the ip with the vm.
5. Volume storage (vs) is created. A step is added to create the vs, and if required a step is added to attach the vs to the vm.

Execute Workload

One you have reviewed the workload plan and are happy with it, you can execute the workload plan.

Note: To execute the workload plan you must first enter your payment information into the ComputeNext website. If you do not have payment information you will receive a “403 Forbidden” error.

The process of execution of a workload plan is called a *transaction*.

To execute the workload plan:

```
>node runcws.js execute
execute (execute workload plan)
options: {
  "url": "http://cws.computenext.com/api/workload/b4ee62da-8dff-4a6b-b39f-54acf26a3a6d/execute",
  "method": "put",
  "auth": {
    "user": "63a25f81-15dd-481e-a4d4-c5ed9a80f93a",
    "pass": "<hidden>"
  }
}
### setting current transactionId: 06d7ab5f-f420-4fea-b0bf-456fbc14d884
----- RESULT -----
{
  "workloadStatus": "in-progress",
  "action": "execute",
  "transactionId": "06d7ab5f-f420-4fea-b0bf-456fbc14d884"
}
```

Note that the *transactionId* is returned and that the transaction is “in-progress”.

The workload plan is now in the process of executing. This can take some time.

You can monitor the progress of the workload execution with the *transaction* API (part of the *workload* API). See below.

Running Workload

When the workload plan executes, it will call the instance API to create instances of various types - for example, key pairs (kp), security groups (sg), virtual machines (vm), etc.

All instances created by the transaction are tagged with the *transactionId*.

If you want to find out all instances that were created by the transaction you can query the instance API to list instances with this transactionId (runcws listitx).

Virtual machine (vm) and volume storage (vs) instances are tagged with the *workloadId*.

If you want to find out all instances that are part of the “running workload” you can query the instance API to list instances with this workloadId (runcws listiwl).

We only have vm and vs workload elements in the workload, so only vm and vs instances are considered to be part of the running workload.

Partial Failures and Rollback

If something fails during the execution of a transaction, there is no automatic rollback.

For example, assume your workload specifies two virtual machine workload elements, VM1 and VM2.

Let’s suppose that during the transaction the creation of one of the virtual machines (VM2) fails for some reason. Because the steps to create virtual machines are (usually) done in parallel, one of the virtual machines is created sucessfully (VM1), but the other (VM2) is not. The transaction will go into “failed” status, and will probably indicate a “reason” of “timeout” or “retries” depending on how the failure manifested itself. The transaction/errors (see below) will contain the details about what step failed and why. But, your “running workload” will still have the virtual machine instance (VM1) that succeeded. (The “running workload” is the collection of instances that are tagged with the workloadId of this workload).

You now have two choices. Either you can attempt the activation again, or you can deactivate. If you know why the create of the virtual machine failed and it is something you can correct, you can update the workload element (for VM2) that failed, and re-activate the workload. This will create a new workload plan. That workload plan will detect that you have already got one virtual machine (VM1) in your “running workload”, so it will only generate step(s) to create the one virtual machine (VM2) that failed the first time.

If you want to rollback the creation of the original virtual machine (VM1) then you can deactivate. That will generate a workload plan to delete VM1.

Note that billing is done on the basis of instance activation, and *not* on completion of a successful transaction. If you have a partial activation, you will be charged for those instances that did get created successfully.

Transaction Status

To get the transaction status:

```
>node runcws.js status
status (retrieve transaction status)
options: {
  "url": "http://cws.computenext.com/api/transaction/06d7ab5f-f420-4fea-b0bf-456fbc14d884/status",
  "method": "get",
  "auth": {
    "user": "63a25f81-15dd-481e-a4d4-c5ed9a80f93a",
    "pass": "<hidden>"
  }
}
----- RESULT -----
{
  "transactionId": "06d7ab5f-f420-4fea-b0bf-456fbc14d884",
  "workloadId": "b4ee62da-8dff-4a6b-b39f-54acf26a3a6d",
  "status": "in-progress",
  "started": "2013-12-19T21:51:20.786Z"
}
```

The transaction status can be one of the following values -

- in-progress
- failed

If “failed”, there will be a reason code -

- retries
- timeout
- cancelled

The transaction will fail if any one of the workload steps fails.

retries

If a workload step fails, it will attempt one retry. If the step fails again, then the transaction fails with reason code “retries”.

timeout

A workload step may timeout waiting for the instance to reach some required state. For example, when creating a virtual machine (vm), the required state is “running”. If the instance does not reach the required state within the time allowed the step will fail and hence the transaction will fail with reason code “timeout”.

cancelled

If the transaction is cancelled while in-progress (see below) then the transaction will go into failed status with reason code “cancelled”.

Transaction Steps

As the workload plan is executed, a log is kept of the progress. Each log entry has a *Log Sequence Number* (the LSN) which is simply a sequence number of the log record. When you call the workload API to get the transaction steps, you are getting those log entries that show the start and end for each step.

Note that the log for only one transaction (the last one) for each workload is maintained. You cannot go back and look at previous transactions.

To get the transaction steps:

```
>node runcws.js steps
steps (retrieve transaction steps)
options: {
  "url": "http://cws.computenext.com/api/transaction/06d7ab5f-f420-4fea-b0bf-456fbc14d884/steps",
```

```
  "method": "get",
  "auth": {
    "user": "63a25f81-15dd-481e-a4d4-c5ed9a80f93a",
    "pass": "<hidden>"
  }
}
----- RESULT -----
[
  {
    "lsn": 1,
    "stepId": "0000_kp_create_kp1",
    "timestamp": "2013-12-19T21:51:20.803Z",
    "status": "in-progress"
  },
  {
    "lsn": 8,
    "stepId": "0000_kp_create_kp1",
    "timestamp": "2013-12-19T21:51:22.948Z",
    "status": "completed",
    "elapsedTimeInSeconds": 2.144
  },
  {
    "lsn": 9,
    "stepId": "0001_sg_create_sg1",
    "timestamp": "2013-12-19T21:51:22.961Z",
    "status": "in-progress"
  },
  {
    "lsn": 11,
    "stepId": "0003_sg_create_sg2",
    "timestamp": "2013-12-19T21:51:22.970Z",
    "status": "in-progress"
  },
  {
    "lsn": 18,
    "stepId": "0001_sg_create_sg1",
    "timestamp": "2013-12-19T21:51:24.031Z",
    "status": "completed",
    "elapsedTimeInSeconds": 1.068
  },
  {
    "lsn": 20,
    "stepId": "0003_sg_create_sg2",
    "timestamp": "2013-12-19T21:51:24.039Z",
    "status": "completed",
    "elapsedTimeInSeconds": 1.067
  },
  {
    "lsn": 21,
    "stepId": "0002_sg_update_add_access_sg1",
    "timestamp": "2013-12-19T21:51:24.060Z",
    "status": "in-progress"
  },
  {
    "lsn": 23,
    "stepId": "0004_sg_update_add_access_sg2",
    "timestamp": "2013-12-19T21:51:24.070Z",
    "status": "in-progress"
  },
  {
    "lsn": 31,
    "stepId": "0002_sg_update_add_access_sg1",
    "timestamp": "2013-12-19T21:51:25.224Z",
    "status": "completed",
    "elapsedTimeInSeconds": 1.163
  },
  {
    "lsn": 34,
    "stepId": "0004_sg_update_add_access_sg2",
    "timestamp": "2013-12-19T21:51:26.201Z",
    "status": "completed",
    "elapsedTimeInSeconds": 2.129
```

```
  },
  {
    "lsn": 35,
    "stepId": "0005_vm_create_vm1",
    "timestamp": "2013-12-19T21:51:26.213Z",
    "status": "in-progress"
  }
]
```

The runcws tool gets all the useful log entries for the transaction. For this API method you can also specify a “begin LSN” and an “end LSN” to get just those log records you need.

Each log record can have the following propeties -

- lsn - the Log Sequence Number
- stepId - the workload step id
- timestamp - when this log entry was created
- status - which can be “in-progress”, “completed” or “failed”
- reason - if the status is “failed” - can be “retries”, “timeout” or “cancelled”
- elapsedTimeInSeconds - if the status is “completed” - this is the time taken for the step

You can see that for stepId “0000_kp_create_kp1” we have two log entries, one with LSN=1 with status “in-progress” and one with LSN=8 with status “completed”. The timestamp indicates when the log entry was logged. So this step began at 21:51:20.803, and completed at 21:51:22.948, for an elapsed time of 2.144 seconds.

However - stepId “0005_vm_create_vm1” (LSN=35) has only a log entry for “in-progress” - which means that at the time we did the method call, this step had started but had not yet completed.

Transaction Errors

To get the transaction errors:

```
>node runcws.js errors
errors (retrieve transaction errors)
options: {
  "url": "http://cws.computenext.com/api/transaction/06d7ab5f-f420-4fea-b0bf-456fbc14d884/errors",
  "method": "get",
  "auth": {
    "user": "63a25f81-15dd-481e-a4d4-c5ed9a80f93a",
    "pass": "<hidden>"
  }
}
STATUS: 404
----- RESULT -----
{
  "code": 404,
  "message": "no errors found: transactionId: 06d7ab5f-f420-4fea-b0bf-456fbc14d884",
  "ticket": "f1e8c7dc-13d4-4d42-b999-2574dc6f1c93"
}
```

In this case there are no errors.

If an error is returned, it will include the stepId of the workload step that failed, plus some detailed error information.

Note that it is possible for this call to return an error for a step, but for the overall transaction to succeed. This is because every step attempts a retry on the first error, so the second attempt might succeed.

Transaction Completes

Get the transaction status:

```
>node runcws.js status
status (retrieve transaction status)
options: {
  "url": "http://cws.computenext.com/api/transaction/06d7ab5f-f420-4fea-b0bf-456fbc14d884/status",
  "method": "get",
  "auth": {
    "user": "63a25f81-15dd-481e-a4d4-c5ed9a80f93a",
    "pass": "<hidden>"
  }
}
----- RESULT -----
{
  "transactionId": "06d7ab5f-f420-4fea-b0bf-456fbc14d884",
  "workloadId": "b4ee62da-8dff-4a6b-b39f-54acf26a3a6d",
  "ended": "2013-12-19T21:52:57.516Z",
  "status": "completed",
  "elapsedTimeInSeconds": 96.73
}
```

The transaction status is “completed” and the elapsedTimeInSeconds is given.

Get the steps:

```
>node runcws.js steps
steps (retrieve transaction steps)
options: {
  "url": "http://cws.computenext.com/api/transaction/06d7ab5f-f420-4fea-b0bf-456fbc14d884/steps",
  "method": "get",
  "auth": {
    "user": "63a25f81-15dd-481e-a4d4-c5ed9a80f93a",
    "pass": "<hidden>"
  }
}
----- RESULT -----
[
  {
    "lsn": 1,
    "stepId": "0000_kp_create_kp1",
    "timestamp": "2013-12-19T21:51:20.803Z",
    "status": "in-progress"
  },
  {
    "lsn": 8,
    "stepId": "0000_kp_create_kp1",
    "timestamp": "2013-12-19T21:51:22.948Z",
    "status": "completed",
    "elapsedTimeInSeconds": 2.144
  },
  {
    "lsn": 9,
    "stepId": "0001_sg_create_sg1",
    "timestamp": "2013-12-19T21:51:22.961Z",
    "status": "in-progress"
  },
  {
    "lsn": 11,
    "stepId": "0003_sg_create_sg2",
    "timestamp": "2013-12-19T21:51:22.970Z",
    "status": "in-progress"
  },
  {
    "lsn": 18,
    "stepId": "0001_sg_create_sg1",
    "timestamp": "2013-12-19T21:51:24.031Z",
    "status": "completed",
    "elapsedTimeInSeconds": 1.068
  },
  {
    "lsn": 20,
    "stepId": "0003_sg_create_sg2",
    "timestamp": "2013-12-19T21:51:24.039Z",
    "status": "completed",
```

```
  "elapsedTimeInSeconds": 1.067
},
{
  "lsn": 21,
  "stepId": "0002_sg_update_add_access_sg1",
  "timestamp": "2013-12-19T21:51:24.060Z",
  "status": "in-progress"
},
{
  "lsn": 23,
  "stepId": "0004_sg_update_add_access_sg2",
  "timestamp": "2013-12-19T21:51:24.070Z",
  "status": "in-progress"
},
{
  "lsn": 31,
  "stepId": "0002_sg_update_add_access_sg1",
  "timestamp": "2013-12-19T21:51:25.224Z",
  "status": "completed",
  "elapsedTimeInSeconds": 1.163
},
{
  "lsn": 34,
  "stepId": "0004_sg_update_add_access_sg2",
  "timestamp": "2013-12-19T21:51:26.201Z",
  "status": "completed",
  "elapsedTimeInSeconds": 2.129
},
{
  "lsn": 35,
  "stepId": "0005_vm_create_vm1",
  "timestamp": "2013-12-19T21:51:26.213Z",
  "status": "in-progress"
},
{
  "lsn": 48,
  "stepId": "0005_vm_create_vm1",
  "timestamp": "2013-12-19T21:52:56.507Z",
  "status": "completed",
  "elapsedTimeInSeconds": 90.292
```

```
  }
]
```

You can see that now all steps have completed, and that VM 1 took about 90 seconds to reach the “running” state.

Get the workload:

```
>node runcws.js getwl
getwl (retrieve workload)
options: {
  "url": "http://cws.computenext.com/api/workload/b4ee62da-8dff-4a6b-b39f-54acf26a3a6d",
  "method": "get",
  "auth": {
    "user": "63a25f81-15dd-481e-a4d4-c5ed9a80f93a",
    "pass": "<hidden>"
  }
}
----- RESULT -----
{
  "workloadId": "b4ee62da-8dff-4a6b-b39f-54acf26a3a6d",
  "ownerId": "d00f16e6-7503-4838-a658-8b60594ff285",
  "uniqueName": "hellovm",
  "name": "Hello VM",
  "description": "Workload 'hello world' for one VM",
  "created": "2013-12-19T01:23:27.354Z",
  "updated": "2013-12-19T21:52:57.528Z",
  "metadata": {
    "test": "this is metadata for the entire workload - can be anything",
    "test1": "another line of metadata"
  },
  "elements": [
    {
      "name": "VM 1",
      "uri": "vm/hpcloud/nova/standard.small",
      "parameters": {
        "imageUri": "image/hpcloud/nova/ami-00000075",
        "keyPair": "KP 1",
        "securityGroups": [
          "SG 1",
          "SG 2"
        ]
      },
      "metadata": {
        "description": "hello world - my first virtual machine"
      }
    }
  ],
  "execute": {
    "created": "2013-12-19T21:52:57.527Z",
    "action": "activate",
    "transactionId": "06d7ab5f-f420-4fea-b0bf-456fbc14d884",
    "elements": [
      {
        "name": "VM 1",
        "uri": "vm/hpcloud/nova/standard.small",
        "parameters": {
          "imageUri": "image/hpcloud/nova/ami-00000075",
          "keyPairId": "*0000_kp_create_kp1",
          "securityGroupIds": [
            "*0001_sg_create_sg1",
            "*0003_sg_create_sg2"
          ]
        },
        "metadata": {
          "description": "hello world - my first virtual machine",
          "name": "VM 1"
        },
        "resource": {
          "id": "vm_hpcloud_nova_standard-small",
          "uri": "vm/hpcloud/nova/standard.small",
          "resourceType": "vm",
          "provider": "hpcloud",
          "region": "nova",
```

```
        "providerResourceId": "Standard.small",
        "cpuSpeed": "1.2",
        "cpuCount": "2",
        "localStorage": "60",
        "ram": "2",
        "operatingSystemVersion": "64 Bit",
        "zone": "nova",
        "connectorType": "openStack.compute"
      }
    }
  ]
},
```

```
  "workloadStatus": "none"
}
```

Note that after the transaction has completed, the plan is no longer in the workload - it is no longer needed. However, if the transaction fails, then the failed plan is included in the *execute* section (see below).

An *execute* section has been added to the workload. This is a snapshot of what the state of the workload was at the last time it was executed.

Get the “running workload” - all the instances that were created for this workload:

```
>node runcws.js listiwl
listiwl (retrieve all instances in the workload)
options: {
  "url": "http://cws.computenext.com/api/instance?workloadId=b4ee62da-8dff-4a6b-b39f-54acf26a3a6d",
  "method": "get",
  "auth": {
    "user": "63a25f81-15dd-481e-a4d4-c5ed9a80f93a",
    "pass": "<hidden>"
  }
}
----- RESULT -----
[
  {
    "instanceId": "dd13e743-33be-4b10-b740-99d8cf4f8d71",
    "created": "2013-12-19T21:51:29.681Z",
    "updated": "2013-12-19T21:52:26.831Z",
    "ownerId": "d00f16e6-7503-4838-a658-8b60594ff285",
    "resourceUri": "vm/hpcloud/nova/standard.small",
    "resourceType": "vm",
    "provider": "hpcloud",
    "region": "nova",
    "providerResourceId": "Standard.small",
    "attributes": {
      "providerInstanceId": 2714865,
      "password": "qeh46YnWszk3aRxa",
      "instanceStatus": "running",
      "transientStatus": false,
      "privateIpAddress": "10.2.252.25",
      "publicIpAddress": "15.185.216.105"
    },
    "attributeTimestamps": {
      "password": "2013-12-19T21:52:26.800Z",
      "instanceStatus": "2013-12-19T21:52:26.809Z",
      "privateIpAddress": "2013-12-19T21:52:26.801Z",
      "publicIpAddress": "2013-12-19T21:52:26.808Z"
    },
    "metadata": {
      "description": "hello world - my first virtual machine",
      "name": "VM 1",
      "workloadId": "b4ee62da-8dff-4a6b-b39f-54acf26a3a6d",
      "transactionId": "06d7ab5f-f420-4fea-b0bf-456fbc14d884"
    },
    "parameters": {
      "imageUri": "image/hpcloud/nova/ami-00000075",
      "keyPairId": "60e3e8e9-08aa-4934-90e2-0002de271fdc",
      "securityGroupIds": [
        "9a200d98-31ea-4bbc-a264-b83e5d6219f6",
        "b1092157-703a-4f30-9ed6-895f3277bcf5"
      ],
      "vm_providerResourceId": "Standard.small",
      "zone": "nova",
      "cpuCount": "2",
      "cpuSpeed": "1.2",
      "localStorage": "60",
      "ram": "2",
      "username": "ubuntu",
      "image_providerResourceId": "ami-00000075",
      "keyPairId_providerInstanceId": "60e3e8e9-08aa-4934-90e2-0002de271fdc",
      "securityGroupIds_providerInstanceId": [
        568643,
        568641
      ]
    }
  }
]
```

Cancel Transaction

We start with a workload that has two virtual machines defined (VM 1 and VM2):

```
>node runcws.js getwl
getwl (retrieve workload)
options: {
  "url": "http://cws.computenext.com/api/workload/b4ee62da-8dff-4a6b-b39f-54acf26a3a6d",
  "method": "get",
  "auth": {
    "user": "63a25f81-15dd-481e-a4d4-c5ed9a80f93a",
    "pass": "<hidden>"
  }
}
----- RESULT -----
{
  "workloadId": "b4ee62da-8dff-4a6b-b39f-54acf26a3a6d",
  "ownerId": "d00f16e6-7503-4838-a658-8b60594ff285",
  "uniqueName": "hellovm",
  "name": "Hello VM",
  "description": "Workload 'hello world' for one VM",
  "created": "2013-12-19T01:23:27.354Z",
  "updated": "2013-12-20T00:04:11.879Z",
  "metadata": {
    "test": "this is metadata for the entire workload - can be anything",
    "test1": "another line of metadata"
  },
  "elements": [
    {
      "name": "VM 1",
      "uri": "vm/hpcloud/nova/standard.small",
      "parameters": {
        "imageUri": "image/hpcloud/nova/ami-00000075",
        "keyPair": "KP 1",
        "securityGroups": [
          "SG 1",
          "SG 2"
        ]
      },
      "metadata": {
        "description": "hello world - my first virtual machine"
      }
    },
    {
      "name": "VM 2",
      "uri": "vm/hpcloud/nova/standard.small",
      "parameters": {
        "imageUri": "image/hpcloud/nova/ami-00000075",
        "keyPair": "KP 1",
        "securityGroups": [
          "SG 1",
          "SG 2"
        ]
      },
      "metadata": {
        "description": "hello world - my SECOND virtual machine"
      }
    }
  ],
  "execute": {
    "created": "2013-12-20T00:03:12.487Z",
    "action": "deactivate",
    "transactionId": "f1e0ec0c-b871-4b71-be30-03eb05b61191",
    "elements": [
      {
        "name": "VM 1",
        "uri": "vm/hpcloud/nova/standard.small",
        "parameters": {
          "imageUri": "image/hpcloud/nova/ami-00000075",
          "keyPair": "KP 1",
          "securityGroups": [
            "SG 1",
            "SG 2"
          ]
        },
        "metadata": {
```

```
          "description": "hello world - my first virtual machine"
        }
      }
    ]
  },
  "workloadStatus": "none"
}
```

Plan activation and then execute:

```
>node runcws.js activate
activate (plan workload activation)
options: {
  "url":
"http://cws.computenext.com/api/workload/b4ee62da-8dff-4a6b-b39f-54acf26a3a6d/plan?action=activate",
  "method": "put",
  "auth": {
    "user": "63a25f81-15dd-481e-a4d4-c5ed9a80f93a",
    "pass": "<hidden>"
  }
}
----- RESULT -----
{
  "workloadStatus": "in-progress",
  "action": "plan-activate"
}

>node runcws.js execute
execute (execute workload plan)
options: {
  "url": "http://cws.computenext.com/api/workload/b4ee62da-8dff-4a6b-b39f-54acf26a3a6d/execute",
  "method": "put",
  "auth": {
    "user": "63a25f81-15dd-481e-a4d4-c5ed9a80f93a",
    "pass": "<hidden>"
  }
}
### setting current transactionId: 654a5e87-04ff-439c-bd51-f08bb033e580
----- RESULT -----
{
  "workloadStatus": "in-progress",
  "action": "execute",
  "transactionId": "654a5e87-04ff-439c-bd51-f08bb033e580"
}
```

Get the workload steps:

```
>node runcws.js steps
steps (retrieve transaction steps)
options: {
  "url": "http://cws.computenext.com/api/transaction/654a5e87-04ff-439c-bd51-f08bb033e580/steps",
  "method": "get",
  "auth": {
    "user": "63a25f81-15dd-481e-a4d4-c5ed9a80f93a",
    "pass": "<hidden>"
  }
}
----- RESULT -----
[
  {
    "lsn": 1,
    "stepId": "0000_vm_create_vm1",
    "timestamp": "2013-12-20T00:04:44.651Z",
    "status": "in-progress"
  },
  {
    "lsn": 3,
    "stepId": "0001_vm_create_vm2",
    "timestamp": "2013-12-20T00:04:44.659Z",
    "status": "in-progress"
  }
]
```

The create of both virtual machines is in-progress.

Cancel the transaction:

```
>node runcws.js cancel
cancel (cancel transaction)
options: {
  "url": "http://cws.computenext.com/api/transaction/654a5e87-04ff-439c-bd51-f08bb033e580/cancel",
  "method": "put",
  "auth": {
    "user": "63a25f81-15dd-481e-a4d4-c5ed9a80f93a",
    "pass": "<hidden>"
  }
}
----- RESULT -----
{
  "transactionId": "654a5e87-04ff-439c-bd51-f08bb033e580",
  "workloadId": "b4ee62da-8dff-4a6b-b39f-54acf26a3a6d",
  "action": "cancel",
  "status": "in-progress"
}
```

The cancel operation is in-progress.

If we poll with “steps” eventually we see:

```
>node runcws.js steps
steps (retrieve transaction steps)
options: {
  "url": "http://cws.computenext.com/api/transaction/654a5e87-04ff-439c-bd51-f08bb033e580/steps",
  "method": "get",
  "auth": {
    "user": "63a25f81-15dd-481e-a4d4-c5ed9a80f93a",
    "pass": "<hidden>"
  }
}
----- RESULT -----
[
  {
    "lsn": 1,
    "stepId": "0000_vm_create_vm1",
    "timestamp": "2013-12-20T00:04:44.651Z",
    "status": "in-progress"
  },
  {
    "lsn": 3,
    "stepId": "0001_vm_create_vm2",
    "timestamp": "2013-12-20T00:04:44.659Z",
    "status": "in-progress"
  },
  {
    "lsn": 7,
    "stepId": "0001_vm_create_vm2",
    "timestamp": "2013-12-20T00:05:14.802Z",
    "status": "failed",
    "reason": "cancelled",
    "elapsedTimeInSeconds": 30.14
  },
  {
    "lsn": 8,
    "stepId": "0000_vm_create_vm1",
    "timestamp": "2013-12-20T00:05:14.810Z",
    "status": "failed",
    "reason": "cancelled",
    "elapsedTimeInSeconds": 30.157
  }
]
```

Both steps have been cancelled.

If we poll with “status”, eventually we see:

```
>node runcws.js status
status (retrieve transaction status)
options: {
  "url": "http://cws.computenext.com/api/transaction/654a5e87-04ff-439c-bd51-f08bb033e580/status",
  "method": "get",
  "auth": {
    "user": "63a25f81-15dd-481e-a4d4-c5ed9a80f93a",
    "pass": "<hidden>"
  }
}
----- RESULT -----
{
  "transactionId": "654a5e87-04ff-439c-bd51-f08bb033e580",
  "workloadId": "b4ee62da-8dff-4a6b-b39f-54acf26a3a6d",
  "ended": "2013-12-20T00:05:59.814Z",
  "status": "failed",
  "reason": "cancelled",
  "stepId": "0001_vm_create_vm2",
  "elapsedTimeInSeconds": 75.174
}
```

Note that the transaction status may take some time to reach “cancelled” after the steps have reached “cancelled”.

The first step that detected that the workload was cancelled is identified.

However: even though we cancelled the transaction, the requests to create the virtual machines had already been sent to the region. So the virtual machine instances are actually created.

Get the “running workload”:

```
>node runcws.js listiwl
listiwl (retrieve all instances in the workload)
options: {
  "url": "http://cws.computenext.com/api/instance?workloadId=b4ee62da-8dff-4a6b-b39f-54acf26a3a6d",
  "method": "get",
  "auth": {
    "user": "63a25f81-15dd-481e-a4d4-c5ed9a80f93a",
    "pass": "<hidden>"
  }
}
----- RESULT -----
[
  {
    "instanceId": "0523e9cf-0fe9-4b8d-91e2-21aeca5a1ae3",
    "created": "2013-12-20T00:04:53.328Z",
    "updated": "2013-12-20T00:04:53.328Z",
    "ownerId": "d00f16e6-7503-4838-a658-8b60594ff285",
    "resourceUri": "vm/hpcloud/nova/standard.small",
    "resourceType": "vm",
    "provider": "hpcloud",
    "region": "nova",
    "attributes": {
      "providerInstanceId": 2715437,
      "password": "AFdTwcwmFQc5DyAx",
      "instanceStatus": "creating",
      "transientStatus": true
    },
    "attributeTimestamps": {
      "password": "2013-12-20T00:04:53.318Z",
      "instanceStatus": "2013-12-20T00:04:53.325Z"
    },
    "metadata": {
      "description": "hello world - my first virtual machine",
      "name": "VM 1",
      "workloadId": "b4ee62da-8dff-4a6b-b39f-54acf26a3a6d",
      "transactionId": "654a5e87-04ff-439c-bd51-f08bb033e580"
    },
    "parameters": {
      "imageUri": "image/hpcloud/nova/ami-00000075",
      "keyPairId": "60e3e8e9-08aa-4934-90e2-0002de271fdc",
      "securityGroupIds": [
        "9a200d98-31ea-4bbc-a264-b83e5d6219f6",
```

```
      "b1092157-703a-4f30-9ed6-895f3277bcf5"
    ],
    "vm_providerResourceId": "Standard.small",
    "zone": "nova",
    "cpuCount": "2",
    "cpuSpeed": "1.2",
    "localStorage": "60",
    "ram": "2",
    "username": "ubuntu",
    "image_providerResourceId": "ami-00000075",
    "keyPairId_providerInstanceId": "60e3e8e9-08aa-4934-90e2-0002de271fdc",
    "securityGroupIds_providerInstanceId": [
      568643,
      568641
    ]
  }
},
{
  "instanceId": "b6011557-1685-4379-856f-3cec844887fc",
  "created": "2013-12-20T00:04:53.353Z",
  "updated": "2013-12-20T00:04:53.353Z",
  "ownerId": "d00f16e6-7503-4838-a658-8b60594ff285",
  "resourceUri": "vm/hpcloud/nova/standard.small",
  "resourceType": "vm",
  "provider": "hpcloud",
  "region": "nova",
  "attributes": {
    "providerInstanceId": 2715435,
    "password": "ykcLunMnGfH66Coz",
    "instanceStatus": "creating",
    "transientStatus": true
  },
  "attributeTimestamps": {
    "password": "2013-12-20T00:04:53.347Z",
    "instanceStatus": "2013-12-20T00:04:53.351Z"
  },
  "metadata": {
    "description": "hello world - my SECOND virtual machine",
    "name": "VM 2",
    "workloadId": "b4ee62da-8dff-4a6b-b39f-54acf26a3a6d",
    "transactionId": "654a5e87-04ff-439c-bd51-f08bb033e580"
  },
  "parameters": {
    "imageUri": "image/hpcloud/nova/ami-00000075",
    "keyPairId": "60e3e8e9-08aa-4934-90e2-0002de271fdc",
    "securityGroupIds": [
      "9a200d98-31ea-4bbc-a264-b83e5d6219f6",
      "b1092157-703a-4f30-9ed6-895f3277bcf5"
    ],
    "vm_providerResourceId": "Standard.small",
    "zone": "nova",
    "cpuCount": "2",
    "cpuSpeed": "1.2",
    "localStorage": "60",
    "ram": "2",
    "username": "ubuntu",
    "image_providerResourceId": "ami-00000075",
    "keyPairId_providerInstanceId": "60e3e8e9-08aa-4934-90e2-0002de271fdc",
    "securityGroupIds_providerInstanceId": [
      568643,
      568641
    ]
  }
```

```
  }
]
```

Therefore you always need to check what *running instances* you actually have for the workload, because the transaction may have created one or more instances even though the transaction itself has failed or been cancelled. And as noted before, billing considers the instances for charging purposes, not the transactions or workloads.

If you want to perform a “rollback” of all vm and vs instances created by the transaction then you can do a deactivation - see next section.

Plan Workload Deactivation

To generate a workload plan for deactivation:

```
>node runcws.js deactivate
deactivate (plan workload deactivation)
options: {
  "url":
"http://cws.computenext.com/api/workload/b4ee62da-8dff-4a6b-b39f-54acf26a3a6d/plan?action=deactivate",
  "method": "put",
  "auth": {
    "user": "63a25f81-15dd-481e-a4d4-c5ed9a80f93a",
    "pass": "<hidden>"
  }
}
----- RESULT -----
{
  "workloadStatus": "in-progress",
  "action": "plan-deactivate"
}
```

Get the workload:

```
>node runcws.js getwl
getwl (retrieve workload)
options: {
  "url": "http://cws.computenext.com/api/workload/b4ee62da-8dff-4a6b-b39f-54acf26a3a6d",
  "method": "get",
  "auth": {
    "user": "63a25f81-15dd-481e-a4d4-c5ed9a80f93a",
    "pass": "<hidden>"
  }
}
----- RESULT -----
{
  "workloadId": "b4ee62da-8dff-4a6b-b39f-54acf26a3a6d",
  "ownerId": "d00f16e6-7503-4838-a658-8b60594ff285",
  "uniqueName": "hellovm",
  "name": "Hello VM",
  "description": "Workload 'hello world' for one VM",
  "created": "2013-12-19T01:23:27.354Z",
  "updated": "2013-12-19T21:52:57.528Z",
  "metadata": {
    "test": "this is metadata for the entire workload - can be anything",
    "test1": "another line of metadata"
  },
  "elements": [
    {
      "name": "VM 1",
      "uri": "vm/hpcloud/nova/standard.small",
      "parameters": {
        "imageUri": "image/hpcloud/nova/ami-00000075",
        "keyPair": "KP 1",
        "securityGroups": [
          "SG 1",
          "SG 2"
        ]
      },
      "metadata": {
        "description": "hello world - my first virtual machine"
      }
    }
```

```
],
"execute": {
  "created": "2013-12-19T21:52:57.527Z",
  "action": "activate",
  "transactionId": "06d7ab5f-f420-4fea-b0bf-456fbc14d884",
  "elements": [
    {
      "name": "VM 1",
      "uri": "vm/hpcloud/nova/standard.small",
      "parameters": {
        "imageUri": "image/hpcloud/nova/ami-00000075",
        "keyPairId": "*0000_kp_create_kp1",
        "securityGroupIds": [
          "*0001_sg_create_sg1",
          "*0003_sg_create_sg2"
        ]
      },
      "metadata": {
        "description": "hello world - my first virtual machine",
        "name": "VM 1"
      },
      "resource": {
        "id": "vm_hpcloud_nova_standard-small",
        "uri": "vm/hpcloud/nova/standard.small",
        "resourceType": "vm",
        "provider": "hpcloud",
        "region": "nova",
        "providerResourceId": "Standard.small",
        "cpuSpeed": "1.2",
        "cpuCount": "2",
        "localStorage": "60",
        "ram": "2",
        "operatingSystemVersion": "64 Bit",
        "zone": "nova",
        "connectorType": "openStack.compute"
      }
    }
  ]
},
"workloadStatus": "none",
"plan": {
  "action": "deactivate",
  "ownerId": "d00f16e6-7503-4838-a658-8b60594ff285",
  "created": "2013-12-19T22:54:48.108Z",
  "expires": "2013-12-19T22:59:48.108Z",
  "elements": [
    {
      "name": "VM 1",
      "uri": "vm/hpcloud/nova/standard.small",
      "parameters": {
        "imageUri": "image/hpcloud/nova/ami-00000075",
        "keyPair": "KP 1",
        "securityGroups": [
          "SG 1",
          "SG 2"
        ]
      },
      "metadata": {
        "description": "hello world - my first virtual machine"
      }
    }
  ],
  "serial": [
    {
      "parallel": [
        {
          "step": {
            "id": "0000_vm_delete_vm1",
            "action": "vm.delete",
            "instanceId": "dd13e743-33be-4b10-b740-99d8cf4f8d71",
            "uri": "vm/hpcloud/nova/standard.small",
            "resource": {
```

```
              "provider": "hpcloud",
              "region": "nova",
              "resourceType": "vm"
            },
            "metadata": {
              "description": "hello world - my first virtual machine",
              "name": "VM 1",
              "transactionId": "06d7ab5f-f420-4fea-b0bf-456fbc14d884",
              "originalWorkloadId": "b4ee62da-8dff-4a6b-b39f-54acf26a3a6d"
            },
            "timing": {
              "min": 15.08,
              "avg": 15.08,
              "max": 15.08
            }
          }
        }
      ]
    }
  ]
}
```


You can see that a step has been added to delete the virtual machine instance.

You can now go ahead and execute this plan (as before, for activate) and it will delete the vm instance.