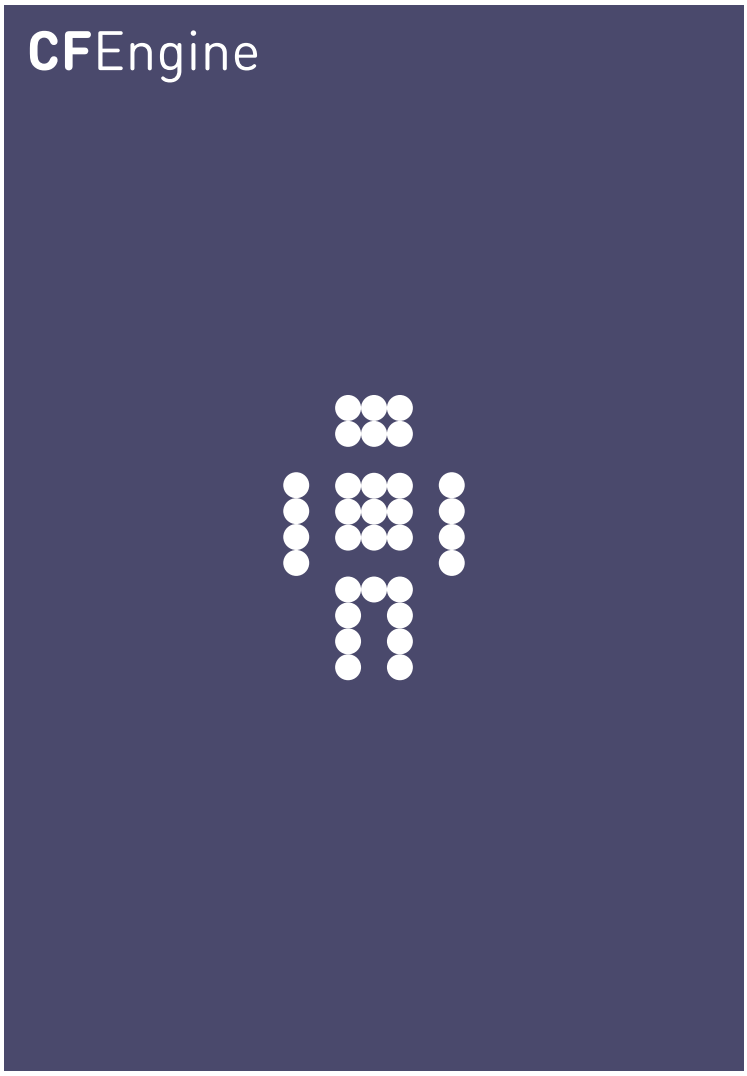

CFEngine Enterprise 3.0 API

CFEngine Enterprise Documentation

Updated 23. November 2012

CFEngine AS

Table of Contents

1 CFEngine Enterprise 3.0 API

The CFEngine Enterprise API allows HTTP clients to interact with the Hub of a CFEngine Enterprise 3.0 installation. With the Enterprise API, you can..

- Check installation status
- Manage users, groups and settings
- Browse host (agent) information and policy
- Issue flexible SQL queries against data collected by the Hub from agents
- Schedule reports for email and later download

The Enterprise API is a REST API, but a central part of interacting with the API involves using SQL. This is new in 3.0 and was done to provide users with maximal flexibility for crafting custom reports based on the wealth of data residing on the Hub.

1.1 Basic Properties of the API

1.1.1 HTTP + JSON

The Enterprise API is a conventional REST API in the sense that it has a number of URI resources that support one or more GET, PUT, POST, or DELETE operations. While reporting is done using SQL, this query is always wrapped in a JSON request.

1.1.2 Requests

GET requests are one of **listing** or **getting**. **Listing** resources means that a number of results will be returned, but each entry may contain limited information. An example of a **listing** query is */api/user* to list users. Notice that URI components are always non-plural. An exception to this is */api/settings*, which returns the singleton resource for settings. **Getting** a resource specifies an individual resource to return, e.g. */api/user/homer*. **PUT** request typically create a new resource, e.g. a user. **POST** requests typically updates an existing resource. **DELETE** requests are also supported in some cases.

1.1.3 Responses

Enterprise 3.0 API responses are always of the following format, consisting of a **meta** object and a **data** array.

```
{
  "meta": {
    "page": 1,
    "count": 1,
    "total": 1,
    "timestamp": 1350922925
  },
  "data": [
     ...
  ]
}
```

page refers to the current page number of the request. **count** is the number of results in the current page, equaling the length of the **data** array. **total** is the number of results in all available pages

combined. **timestamp** is the time the request was processed by the API. The **data** array is resource dependent, but will always contain objects. Response objects typically do not contain error codes. If the response is not *200 OK*, the appropriate HTTP error code returned along with a (possibly non-JSON) payload.

1.1.4 Pagination

Pagination is handled by **page** and **count** query parameters to a **GET** request, e.g. */api/user?page=5&count=30* to get the 5th page of pages of 30 entries each. The default **page** is 1 and the default **count** is 50 if these are not specified explicitly.

1.1.5 Time

All timestamps are reported in *Unix Time*, i.e. seconds since 1970.

1.1.6 Authentication

The API supports both internal and external authentication. The internal users table will always be consulted first, followed by an external source specified in the settings. External sources are *OpenLDAP* or *Active Directory* servers configurable through *POST /api/settings*.

1.1.7 Authorization

Some resources require that the request user is a member of the *admin* role. Roles are managed with */api/role*. ⟨undefined⟩ [Role Based Access Control (RBAC)], page ⟨undefined⟩ is configurable through settings. Users typically have permission to access their own resources, e.g. their own scheduled reports.

1.2 Differences between the CFEngine Nova 2.2 REST API and the CFEngine Enterprise 3.0 API

1.2.1 Read vs. Read/Write

The 2.2 API was read-only and users, roles and settings was managed by the Mission Portal. By contrast, the 3.0 API is read/write and completely standalone from the Mission Portal. In the CFEngine Enterprise 3.0, users, roles and settings belong in the API, and the Mission Portal uses this to determine access to data. Additionally, some other resources support PUT, POST and DELETE, but most data collected from agents are read-only.

1.2.2 Built-in Reports vs. Reporting Engine

The 2.2 API provided an almost one-to-one correspondence between the reports in the Mission Portal and the API. One of the big changes in CFEngine Enterprise 3.0 is the advent of SQL reports. This is provided to the Mission Portal through the API, and you can use it too. You may issue both synchronous and asynchronous reporting requests, and optionally schedule reports to be received by email.

1.2.3 Content-Type

The 2.2 API has a HTTP content-type `application/vnd.cfengine.nova-v1+json`. In the 3.0 API the content-type is `application/vnd.cfengine.enterprise-v1+json`. This reflects a branding change away from Nova to Enterprise.

1.2.4 New Users

The 2.2 API used credentials from the Mission Portal database to authenticate and authorize users. These users have been moved into the Hub database and security has been strengthened. We are now using salted SHA256 passwords for the user table. Unfortunately, this means that internal users need to be recreated. The Mission Portal now relies on the API for authentication and authorization. This was partially done to support multi-hub installations.

1.2.5 Base Path

The 2.2 API had a base path `/rest`. In the 3.0 API the base path is `/api`.

1.2.6 Still available

In 3.0, the old 2.2 API is still available along side the new 3.0 API, so you can keep calling the old API if needed.

1.2.7 Mission Portal

Starting in 3.0, most of the API is exercised by the Mission Portal web-UI.

1.3 Checking Status

You can get basic info about the API by issuing *GET /api*. This status information may also be useful if you contact support, as it gives some basic diagnostics.

Request

```
curl -k --user admin:admin https://test.cfengine.com/api/
```

Response


```
{
  "meta": {
    "page": 1,
    "count": 1,
    "total": 1,
    "timestamp": 1351154889
  },
  "data": [
    {
      "apiName": "CFEngine Enterprise API",
      "apiVersion": "v1",
      "enterpriseVersion": "3.0.0a1.81c0d4c",
      "coreVersion": "3.5.0a1.f3649b2",
      "databaseHostname": "127.0.0.1",
      "databasePort": 27017,
      "authenticated": "internal",
      "license": {
        "expires": 1391036400,
        "installTime": 1329578143,
        "owner": "Stage Environment",
        "granted": 20,
        "licenseUsage": {
```

```
          "lastMeasured": 1351122120,
          "samples": 1905,
          "minObservedLevel": 7,
          "maxObservedLevel": 30,
          "meanUsage": 21.9689,
          "meanCumulativeUtilization": 109.8446,
          "usedToday": 7
        }
      }
    }
  ]
}
```


1.4 Managing Settings

Most of the settings configurable in the API relate to LDAP authentication of users. Settings support two operations, **GET** (view settings) and **POST** (update settings). When settings are updated, they are sanity checked individually and as a whole. All or no settings will be updated for a request.

1.4.1 Viewing settings

Request

```
curl --user admin:admin http://test.cfengine.com/api/settings
```

Response


```
{
  "meta": {
    "page": 1,
    "count": 1,
    "total": 1,
    "timestamp": 1350992335
  },
  "data": [
    {
      "rbacEnabled": true,
      "ldapEnabled": false,
      "ldapActiveDirectoryDomain": "ad.cfengine.com",
      "ldapBaseDN": "DC=ad,DC=cfengine,DC=com",
      "ldapEncryption": "plain",
      "ldapHost": "ldap-server.cfengine.com",
      "ldapLoginAttribute": "sAMAccountName",
      "ldapMode": "activeDirectory",
      "ldapPassword": "password",
      "ldapPort": 389,
      "ldapPortSSL": 636,
      "ldapUsername": "test",
      "ldapUsersDirectory": "CN=Users",
      "blueHostHorizon": 900,
```

```
      "logLevel": "error"
    }
  ]
}
```

1.4.2 Example: Configuring LDAP

The setting **ldapEnabled** turns external authentication on or off. When turned on, the API will check to see that the other LDAP related settings make sense, and attempt to authenticate using the configured credentials. If it is not successful in doing this, no settings will be changed. The API will notify you with a return code an a message describing the error.

Request

```
curl --user admin:admin http://test.cfengine.com/api/settings -X POST -d
{
  "ldapEnabled": true,
  "ldapActiveDirectoryDomain": "ad.cfengine.com",
  "ldapBaseDN": "DC=ad,DC=example,DC=com",
  "ldapEncryption": "ssl",
  "ldapHost": "ldap-server.cfengine.com",
  "ldapLoginAttribute": "sAMAccountName",
  "ldapMode": "standard",
  "ldapPassword": "password",
  "ldapUsername": "test",
  "ldapUsersDirectory": "ou",
}
```

Response

```
204 No Content
```

1.4.3 Example: Configuring Active Directory

Active Directory is configured in much the same way as OpenLDAP, but the additional field **ldapActiveDirectoryDomain** is required. **ldapMode** is also changed from *standard* to *activeDirectory*.

Request

```
curl --user admin:admin http://test.cfengine.com/api/settings -X POST -d
{
  "ldapEnabled": true,
  "ldapBaseDN": "DC=example,DC=com",
  "ldapEncryption": "plain",
  "ldapHost": "ad-server.cfengine.com",
  "ldapLoginAttribute": "uid",
  "ldapMode": "activeDirectory",
  "ldapPassword": "password",
  "ldapUsername": "test",
  "ldapUsersDirectory": "CN=Users",
}
```

Response

```
204 No Content
```

1.4.4 Example: Changing The Log Level

The API uses standard Unix syslog to log a number of events. Additionally, log events are sent to stderr, which means they may also end up in your Apache log. Log events are filtered based on the log level in settings. Suppose you wanted to have greater visibility into the processing done at the backend. The standard log level is *error*. Changing it to *info* is done as follows.

Request

```
curl --user admin:admin http://test.cfengine.com/api/settings -X POST -d
{
  "logLevel": "info"
}
```

Response

```
204 No Content
```

1.5 Managing Users and Roles

Users and Roles determine who has access to what data from the API. Roles are defined by regular expressions that determine which hosts the user can see, and what policy outcomes are restricted.

1.5.1 Example: Listing Users

Request

```
curl --user admin:admin http://test.cfengine.com/api/user
```

Response

```
{
  "meta": {
    "page": 1,
    "count": 2,
    "total": 2,
    "timestamp": 1350994249
  },
  "data": [
    {
      "id": "calvin",
      "external": true,
      "roles": [
        "Huguenots", "Marketing"
      ]
    },
    {
      "id": "quinester",
```

```
      "name": "Willard Van Orman Quine",
      "email": "noreply@@aol.com",
      "external": false,
      "roles": [
        "admin"
      ]
    }
  ]
}
```

1.5.2 Example: Creating a New User

All users will be created for the internal user table. The API will never attempt to write to an external LDAP server.

Request

```
curl --user admin:admin http://test.cfengine.com/api/user/snookie -X PUT -d
{
  "email": "snookie@mtv.com",
  "roles": [
    "HR"
  ]
}
```

Response

```
201 Created
}
```

1.5.3 Example: Updating an Existing User

Both internal and external users may be updated. When updating an external users, the API will essentially annotate metadata for the user, it will never write to LDAP. Consequently, passwords may only be updated for internal users. Users may only update their own records, as authenticated by their user credentials.

Request

```
curl --user admin:admin http://test.cfengine.com/api/user/calvin -X POST -d
{
  "name": "Calvin",
}
```

Response

```
204 No Content
}
```

1.5.4 Example: Retrieving a User

It is possible to retrieve data on a single user instead of listing everything. The following query is similar to issuing *GET /api/user?id=calvin*, with the exception that the previous query accepts a regular expression for *id*.

Request

```
curl --user admin:admin http://test.cfengine.com/api/user/calvin
```

Response

```
{
  "meta": {
    "page": 1,
    "count": 1,
    "total": 1,
    "timestamp": 1350994249
  },
  "data": [
    {
      "id": "calvin",
      "name": "Calvin",
      "external": true,
      "roles": [
        "Huguenots", "Marketing"
      ]
    },
  ]
}
```

1.5.5 Example: Adding a User to a Role

Adding a user to a role is just an update operation on the user. The full role-set is updated, so if you are only appending a role, you may want to fetch the user data first, append the role and then update. The same approach is used to remove a user from a role.

Request

```
curl --user admin:admin http://test.cfengine.com/api/user/snookie -X POST -d
{
  "roles": [
    "HR", "gcc-contrib"
  ]
}
```

Response

```
204 No Content
}
```

1.5.6 Example: Deleting a User

Users can only be deleted from the internal users table.

Request

```
curl --user admin:admin http://test.cfengine.com/api/user/snookie -X DELETE
```

Response

```
204 No Content
```

1.5.7 Example: Creating a New Role

Once you've learned how to manage users, managing roles is pretty much the same thing. Roles are defined by four fields that filter host data and policy data: **includeContext**, **excludeContext**, **includeBundles**, **excludeBundles**. Each field is a comma separated list of regular expressions. See the corresponding section on RBAC for an explanation of these fields. Updating, and deleting roles are similar to updating and deleting users, using POST and DELETE.

Request

```
curl --user admin:admin http://test.cfengine.com/api/user/solaris-admins -X PUT -d
{
  "email": "snookie@mtv.com",
  "roles": [
    "description": "Users managing 64-bit Solaris boxes",
    "includeContext": "solaris,x86_64",
  ]
}
```

Response

```
204 No Content
```

1.6 Browsing Host Information

A resource */api/host* is added as an alternative interface for browsing host information. For full flexibility we recommend using SQL reports via */api/query* for this, however, currently vital signs (data gathered from cf-monitord) is not part of the SQL reports data model.

1.6.1 Example: Listing Hosts With A Given Context

Request

```
curl --user admin:admin http://test.cfengine.com/api/host?context-include=windows.*
```

Response

```
{
  "meta": {
    "page": 1,
    "count": 2,
```

```
    "total": 2,
    "timestamp": 1350997528
  },
  "data": [
    {
      "id": "1c8fafe478e05eec60fe08d2934415c81a51d2075aac27c9936e19012d625cb8",
      "hostname": "windows2008-2.test.cfengine.com",
      "ip": "172.20.100.43"
    },
    {
      "id": "dddc95486d97e4308f164ddc1fdbbc133825f35254f9cfbd59393a671015ab99",
      "hostname": "windows2003-2.test.cfengine.com",
      "ip": "172.20.100.42"
    }
  ]
}
```


1.6.2 Example: Looking Up Hosts By Hostname

Contexts are powerful, as you can use them to categorize hosts according to a rich set of tags. For example, each host is automatically tagged with a canonicalized version of its hostname and IP-address. So we could lookup the host with hostname *windows2003-2.test.cfengine.com* as follows (lines split and indented for presentability).

Request

```
curl --user admin:admin http://test.cfengine.com/api/host?context-include=
   windows2003_2_stage_cfengine_com
```

Response


```
{
  "meta": {
    "page": 1,
    "count": 1,
    "total": 1,
    "timestamp": 1350997528
  },
  "data": [
    {
      "id": "dddc95486d97e4308f164ddc1fdbbc133825f35254f9cfbd59393a671015ab99",
      "hostname": "windows2003-2.test.cfengine.com",
      "ip": "172.20.100.42"
    }
  ]
}
```

1.6.3 Example: Looking Up Hosts By IP

Similarly we can lookup the host with hostname *windows2008-2.test.cfengine.com* by IP as follows (lines split and indented for presentability).

Request

```
curl --user admin:admin http://test.cfengine.com/api/host?
   context-include=172_20_100_43
```

Response

```
{
  "meta": {
    "page": 1,
    "count": 1,
    "total": 1,
    "timestamp": 1350997528
  },
  "data": [
    {
      "id": "1c8fafe478e05eec60fe08d2934415c81a51d2075aac27c9936e19012d625cb8",
      "hostname": "windows2008-2.stage.cfengine.com",
      "ip": "172.20.100.43"
    }
  ]
}
```

1.6.4 Example: Removing Host Data

If a host has been decommissioned from a Hub, we can explicitly remove data associated with the host from the Hub, by issuing a DELETE request (lines split and indented for presentability).

Request

```
curl --user admin:admin http://test.cfengine.com/api/host/
   1c8fafe478e05eec60fe08d2934415c81a51d2075aac27c9936e19012d625cb8 -X DELETE
```

Response

```
204 No Content
```

1.6.5 Example: Listing Available Vital Signs For A Host

Each host record on the Hub has a set of vital signs collected by cf-monitord on the agent. We can view the list of vitals signs from as host as follows (lines split and indented for presentability).

Request

```
curl --user admin:admin http://test.cfengine.com/api/host/
   4e913e2f5ccf0c572b9573a83c4a992798cee170f5ee3019d489a201bc98a1a/vital
```

Response

```
{
  "meta": {
    "page": 1,
    "count": 4,
    "total": 4,
    "timestamp": 1351001799
  },
  "data": [
    {
      "id": "messages",
      "description": "New log entries (messages)",
      "units": "entries",
      "timestamp": 1351001400
    },
    {
      "id": "mem_swap",
      "description": "Total swap size",
      "units": "megabytes",
      "timestamp": 1351001400
    },
    {
      "id": "mem_freeswap",
      "description": "Free swap size",
      "units": "megabytes",
      "timestamp": 1351001400
    },
    {
      "id": "mem_free",
      "description": "Free system memory",
      "units": "megabytes",
      "timestamp": 1351001400
    },
}
```


1.6.6 Example: Retrieving Vital Sign Data

Each vital sign has a collected time series of values for up to one week. Here we retrieve the time series for the *mem_free* vital sign at host *4e913e2f5ccf0c572b9573a83c4a992798cee170f5ee3019d489a201bc98a1a* for October 23rd 2012 12:20pm to 12:45pm GMT (lines split and indented for presentability).

Request

```
curl --user admin:admin http://test.cfengine.com/api/host/
   4e913e2f5ccf0c572b9573a83c4a992798cee170f5ee3019d489a201bc98a1a/
   vital/mem_free?from=1350994800&to=1350996300
```

Response


```
"meta": {
```

```
    "page": 1,
    "count": 1,
    "total": 1,
    "timestamp": 1351002265
  },
  "data": [
    {
      "id": "mem_free",
      "description": "Free system memory",
      "units": "megabytes",
      "timestamp": 1351001700,
      "values": [
        [
          1350994800,
          36.2969
        ],
        [
          1350995100,
          36.2969
        ],
        [
          1350995400,
          36.2969
        ],
        [
          1350995700,
          36.2969
        ],
        [
          1350996000,
          36.1758
        ],
        [
          1350996300,
          36.2969
        ]
      ]
    }
  ]
```


1.7 SQL Queries

The standard way of creating reports in CFEngine Enterprise 3.0 is with SQL queries. See the chapter on SQL queries for an explanation. The API has a few ways of creating a report.

- Synchronous query, where we issue a query and wait for the table to be sent back with the response.
- Asynchronous query, where we get a response immediately with an id that we can later query to

download the report.

- Subscribed query, where we specify a query to be run on a schedule and have the result emailed to someone.

1.7.1 Synchronous Queries

Issuing a synchronous query is the most straight forward way of running an SQL query. We simply issue the query and wait for a result to come back.

1.7.1.1 Example: Listing Hostname and IP for Ubuntu Hosts

Request (lines split and indented for presentability)

```
curl -k --user admin:admin https://test.cfengine.com/api/query -X POST -d
{
  "query": "SELECT Hosts.HostName, Hosts.IPAddress FROM Hosts JOIN Contexts
     ON Hosts.Hostkey = Contexts.HostKey WHERE Contexts.ContextName = \"ubuntu\""
}
```

Response

```
{
  "meta": {
    "page": 1,
    "count": 1,
    "total": 1,
    "timestamp": 1351003514
  },
  "data": [
    {
      "query": "SELECT Hosts.HostName, Hosts.IPAddress FROM Hosts JOIN Contexts ON
          Hosts.Hostkey = Contexts.HostKey WHERE Contexts.ContextName = \"ubuntu\"",
      "header": [
        "HostName",
        "IPAddress"
      ],
      "rowCount": 3,
      "rows": [
        [
          "ubuntu10-2.stage.cfengine.com",
          "172.20.100.1"
        ],
        [
          "ubuntu10-3.stage.cfengine.com",
          "172.20.100.2"
        ],
        [
          "ubuntu10-4.stage.cfengine.com",
          "172.20.100.3"
```

```
        ]
      ],
      "cached": false,
      "sortDescending": false
    }
  ]
}
```

The **cached** and **sortDescending** fields here mean that the the result was not retrieved from cache, and that post-processing sorting was not applied. It is also possible to specify **skip** and **limit** fields that will be applied to the result set after it is returned by the SQL engine. These fields are mainly used by the Mission Portal to paginate quickly on already processed queries.

1.7.2 Asynchronous Queries

Because some queries may take some time to compute, it is possible to fire off a query and check the status of it later. This is useful for dumping a lot of data into CSV files for example. The sequence consists of three steps.

1. Issue the asynchronous query and get a job id
2. Check status of processing using the id
3. When the query is completed, get a download link using the id

1.7.2.1 Issuing The Query

Request

```
curl -k --user admin:admin https://test.cfengine.com/api/query/async -X POST -d
{
  "query": "SELECT Hosts.HostName, Hosts.IPAddress FROM Hosts JOIN Contexts ON Hosts.Hostkey = Conte
}
```

Response (lines split and indented for presentability)

```
{
  "meta": {
    "page": 1,
    "count": 1,
    "total": 1,
    "timestamp": 1351003514
  },
  "data": [
    {
      "id": "32ecb0a73e735477cc9b1ea8641e5552",
      "query": "SELECT Hosts.HostName, Hosts.IPAddress FROM Hosts JOIN Contexts ON
          Hosts.Hostkey = Contexts.HostKey WHERE Contexts.ContextName = \"ubuntu\""
    }
  ]
]
```

1.7.2.2 Checking Status

Request

```
curl -k --user admin:admin https://test.cfengine.com/api/query/async/:id
```

Response

```
{
  "meta": {
    "page": 1,
    "count": 1,
    "total": 1,
    "timestamp": 1351003514
  },
  "data": [
    {
      "id": "32ecb0a73e735477cc9b1ea8641e5552",
      "percentageComplete": 42,
    ]
}
```

1.7.2.3 Getting The Completed Report

This is the same API call as checking the status. Eventually, the **percentageComplete** field will reach 100 and there will be a link to the completed report available for downloading.

Request

```
curl -k --user admin:admin https://test.cfengine.com/api/query/async/:id
```

Response

```
{
  "meta": {
    "page": 1,
    "count": 1,
    "total": 1,
    "timestamp": 1351003514
  },
  "data": [
    {
      "id": "32ecb0a73e735477cc9b1ea8641e5552",
      "percentageComplete": 100,
      "href": "https://test.cfengine.com/api/static/32ecb0a73e735477cc9b1ea8641e5552.csv"
    }
  ]
}
```

1.7.3 Subscribed Queries

Subscribed queries happen in the context of a user. Any user can create a query on a schedule and have it email to someone.

1.7.3.1 Example: Creating A Subscribed Query

Here we create a new query to count file changes by name and have the result sent to us by email. The schedule field is any CFEngine context expression. The backend polls subscriptions in a loop and checks whether it's time to generate a report and send it out. In the following example, user *milton* creates a new subscription to a report which he names *file-changes-report*, which will be sent out every Monday night. His boss will get an email with a link to a PDF version of the report.

Request (lines split and indented for presentability)

```
curl -k --user admin:admin https://test.cfengine.com/api/user/milton/
   subscription/query/file-changes-report -X PUT -d
{
  "to": "boss@megaco.com",
  "query": "SELECT Name Count(1) FROM FileChanges GROUP BY Name",
  "schedule": "Monday.Hr23.Min59",
  "title": "A very important file changes report"
  "description": "Text that will be included in email"
  "outputTypes": [ "pdf" ]
}
```

Response

```
204 No Content
```

1.7.3.2 Example: Listing Report Subscriptions

Milton can list all his current subscriptions by issuing the following.

Request

```
curl -k --user admin:admin https://test.cfengine.com/api/user/milton/subscription/query
```

Response

```
{
  "meta": {
    "page": 1,
    "count": 1,
    "total": 1,
    "timestamp": 1351003514
  },
  "data": [
    {
      "id": "file-changes-report"
      "to": "boss@megaco.com",
      "query": "SELECT Name Count(1) FROM FileChanges GROUP BY Name",
      "title": "A very important file changes report"
      "description": "Text that will be included in email"
      "schedule": "Monday.Hr23.Min59",
```

```
      "outputTypes": [ "pdf" ]
    }
]
```

1.7.3.3 Example: Removing A Report Subscription

Request (lines split and indented for presentability)

```
curl -k --user admin:admin https://test.cfengine.com/api/user/milton/
   subscription/query/file-changes-report -X DELETE
```

Response

```
204 No Content
```

1.8 API Reference

1.8.1 /api

Supported Operations:
GET

Fields:

- **apiName** *(string)* Human-friendly API name.
- **apiVersion** *(string)* API version string.
- **enterpriseVersion** *(string)* Version of the CFEngine Enterprise build.
- **coreVersion** *(string)* The version of CFEngine Core (Community) the Enterprise version was built against.
- **databaseHostname** *(string)* Hostname (or IP) of the database the API is connected to.
- **databasePort** *(integer)* Port number of the database the API is connected to.
- **authenticated** *("internal", "external")*, Whether the request was authenticated using the internal users table or an external source.
- **license.expires** *(integer)* Time when the license expires.
- **license.installTime** *(integer)* Time when the license was installed.
- **license.owner** *(string)* The name of the license owner.
- **license.granted** *(integer)* Host number capacity granted by the license.
- **license.licenseUsage.lastMeasured** *(integer)* Time when license usage was last updated.
- **license.licenseUsage.samples** *(integer)* Number of samples collected for license usage.
- **license.licenseUsage.minObservedLevel** *(integer)* Minimum number of observed host licenses in use.
- **license.licenseUsage.minObservedLevel** *(integer)* Maximum number of observed host licenses in use.
- **license.licenseUsage.meanUsage** *(integer)* Average number of observed host licenses in use.
- **license.licenseUsage.meanCumulativeUtilization** *(integer)* (not sure)
- **license.licenseUsage.usedToday** *(integer)* Total number of host licenses observed used today.

1.8.2 /api/settings

Supported Operations:
GET, **POST**

Fields:

- **rbacEnabled** *(boolean)* Whether RBAC is applied to requests.
- **ldapEnabled** *(boolean)* Whether external authentication is activated.
- **activeDirectoryDomain** *(string)* AD domain to use if AD is enabled in **ldapMode**.
- **ldapBaseDN** *(string)* LDAP BaseDN to use for external LDAP requests.
- **ldapEncryption** *("plain", "ssl", "tls")* Type of LDAP binding to establish to external LDAP server. (Default: "plain").
- **ldapHost** *(string)* Hostname of external LDAP server.
- **ldapMode** *("standard", "activeDirectory")* Type of LDAP server to use. "standard" is effectively OpenLDAP. (Default: "standard").
- **ldapLoginAttribute** *(string)* LDAP attribute to use for usernames. (default: "uid").
- **ldapUsername** *(string)* LDAP username.
- **ldapPassword** *(string)* LDAP password.
- **ldapUsersDirectory** *(string)* Attribute and value to qualify the directory in which to look up users, e.g. "ou=people".
- **ldapPort** *(integer)* Port for external LDAP connections not using SSL. (default 389).
- **ldapPort** *(integer)* Port for external LDAP connections using SSL. (default 636).
- **blueHostHorizon** *(integer)* Time interval (seconds) for when to consider a host unreachable. (default 900).
- **logLevel** *("emergency", "alert", "critical", "error", "warning", "notice", "info", "debug")* Syslog filter specifying the severity level at which messages produced by the API should be emitted to syslog and apache.log. (default: error).

1.8.3 /api/user

Supported Operations:
GET

Query Parameters:

- **id** *(regex string)* Regular expression for filtering usernames.
- **external** *("true", "false")* Returns only internal users (false) or only external (true), or all if not specified.

1.8.4 /api/user/:id

Supported Operations:
GET, **PUT**, **POST**, **DELETE**

Fields:

- **id** *(string)* ID of a user.

- **password** *(string)* Password of a user. (Never returned from API).
- **email** *(string)* Email address associated with user.
- **roles** *(array of strings)* Set of IDs of roles a user is in. (Default: empty)
- **external** *(boolean)* Whether or not the user was found externally (LDAP).

1.8.5 /api/role

Supported Operations:
GET

1.8.6 /api/role/:id

Supported Operations:
GET, **PUT**, **POST**, **DELETE**

Fields:

- **id** *(string)* ID of a role.
- **description** *(string)* Arbitrary text describing the role
- **includeContext** *(comma delimited string of regular expression strings)* Includes hosts visible to the users in the role.
- **excludeContext** *(comma delimited string of regular expression strings)* Excludes bundles visible to the users in the role.
- **includeBundles** *(comma delimited string of regular expression strings)* Includes bundles visible to the users in the role.
- **excludeBundles** *(comma delimited string of regular expression strings)* Excludes bundles visible to the users in the role.

1.8.7 /api/host

Supported Operations:
GET

Query Parameters:

- **include-context** *(comma delimited string of regular expression strings)* Includes hosts having context matching the expression.
- **exclude-context** *(comma delimited string of regular expression strings)* Excludes hosts having context matching the expression.

1.8.8 /api/host/:host-id

- **id** *(string)* ID of a host.
- **hostname** *(string)* Hostname of a host.
- **ip** *(string)* IP address of a host.

1.8.9 /api/host/:host-id/context

Supported Operations:
GET

1.8.10 /api/host/:host-id/context/:context-id

Supported Operations:
GET

Fields:

- **id** *(string)* ID of a context (class name)
- **mean** *(real)* Occurrence probability of the context in an agent run.
- **stdv** *(real)* Standard deviation of occurrence probability.
- **timestamp** *(integer)* Last time context was activated on agent.

1.8.11 /api/host/:host-id/vital

Supported Operations:
GET

1.8.11.1 /api/host/:host-id/vital/:vital-id

Supported Operations:
GET

Query Parameters:

- **from** *(integer)* Timestamp marking the start of the interval for which to fetch data. Data is only available going back one week.
- **to** *(integer)* End of data interval to be fetched.

Fields:

- **id** *(string)* ID of vital sign.
- **description** *(string)* Description of vital sign.
- **units** *(string)* Measurement unit of vital sign.
- **timestamp** *(integer)* Timestamp of the last received data point.
- **values** *(array of [t, y], where t is the sample timestamp)* Vital sign data.

1.8.12 /api/promise

Supported Operations:
GET

1.8.13 /api/promise/:promise-id

Supported Operations:
GET

Fields:

- **id** *(string)* Promise handle.
- **type** *(string)* Promise type.
- **promiser** *(string)* Promiser of the promise.

- **promisees** *(array of strings)* A list of promisees of the promise.
- **bundle** *(string)* The bundle this promise belongs to
- **comment** *(string)* Associated comment for the promise.

1.8.14 /api/query

Supported Operations:

POST

Fields:

- **query** *(string)* SQL query string.
- **sortColumn** *(string)* Column on which to sort results. This is applied to the result of the SQL query and can be considered post processing. The Mission Portal uses this to sort cached reports.
- **sortDescending** *(bool)* Apply post-sorting descendingly.
- **skip** *(integer)* Number of results to skip for the processed query. The Mission Portal uses this for pagination on cached results.
- **limit** *(integer)* Limit the number of results in the processed query.

1.8.15 /api/query/async

Supported Operations:

POST

Fields:

- **query** *(string)* SQL query string.
- **id** *(string)* ID of the query job.
- **error** *(string)* Error if anything went wrong.

1.8.16 /api/query/async/:async-query-id

Supported Operations:

GET, **DELETE**

Fields:

- **id** *(string)* ID of the query job.
- **percentageComplete** *(integer)* Processing status for the query.
- **href** *(string)* Download link for the finished report.
- **error** *(string)* Error if anything went wrong.