+++ title = "Chef Infra Server API" draft = false

aliases = ["/api_chef_server.html"]

[menu] [menu.infra] title = "Chef Infra Server API" identifier = "chef_infra/managing_chef_infra_server/api_chef_server.md Chef Infra Server API" parent = "chef_infra/

[edit on GitHub]

The Chef Infra Server API is a REST API that provides access to objects on the Chef Infra Server, including nodes, environments, roles, users, organizations, cookb
RSA public key-pairs.

Requirements

In this article

In this article

The Chef Infra Server API has the following requirements:

- The `Accept` header must be set to `application/json`.
- For `PUT` and `POST` requests, the `Content-Type` header must be set to `application/json`.
- The `X-Chef-Version` header must be set to the version of the Chef Infra Server API that is being used.
- A request must be signed using `Mixlib::Authentication`.
- A request must be well-formatted. The easiest way to ensure a well-formatted request is to use the `Chef::ServerAPI` library.

Authentication Headers

Authentication to the Chef Infra Server occurs when a specific set of HTTP headers are signed using a private key that is associated with the machine from which th
using the public key. Only authorized actions are allowed.

{{< note >}}

Most authentication requests made to the Chef Infra Server are abstracted from the user. Such as when using knife or the Chef Infra Server user interface. In some
to be made more explicitly, but still in a way that does not require authentication headers. In a few cases, such as when using arbitrary Ruby code or cURL, it may b
Server.

{{< /note >}}

Header Format

By default, all hashing is done using SHA-1 and encoded in Base64. Base64 encoding should have line breaks every 60 characters. Each canonical header should b

```
Method:HTTP_METHOD
Hashed Path:HASHED_PATH
X-Ops-Content-Hash:HASHED_BODY
X-Ops-Timestamp:TIME
X-Ops-UserId:USERID
```

where:

- `HTTP_METHOD` is the method used in the API request (`GET`, `POST`, and so on)
- `HASHED_PATH` is the path of the request: `/organizations/NAME/name_of_endpoint`. The `HASHED_PATH` must be hashed using SHA-1 and encoded using Base64
  path is `/`), and must not include a query string.
- The private key must be an RSA key in the SSL `.pem` file format. This signature is then broken into character strings (of not more than 60 characters per line) ar

The Chef Infra Server decrypts this header and ensures its content matches the content of the non-encrypted headers that were in the request. The timestamp of
time. One approach generating the signed headers is to use mixlib-authentication, which is a class-based header signing authentication object similar to the one use

Enable SHA-256

Chef Server versions 12.4.0 and above support signing protocol version 1.3, which adds support for SHA-256 algorithms. It can be enabled on Chef Infra Client via t

```
authentication_protocol_version = '1.3'
```

And on Chef knife via `config.rb`:

```
knife[:authentication_protocol_version] = '1.3'
```

Required Headers

The following authentication headers are required:

Feature	Description
`Accept`	The format in which response data from the Chef Infra Server is provided. This header must be set t

Feature	Description
`Content-Type`	The format in which data is sent to the Chef Infra Server. This header is required for `PUT` and `POST` r
`Host`	The host name (and port number) to which a request is sent. (Port number `80` does not need to be s `api.opscode.com:443`.
`X-Chef-Version`	The version of the Chef Infra Client executable from which a request is made. This header ensures tl
`X-Ops-Authorization-N`	One (or more) 60 character segments that comprise the canonical header. A canonical header is sigı also encoded using Base64. If more than one segment is required, each should be named sequentia where `N` represents the integer used by the last header that is part of the request.
`X-Ops-Content-Hash`	The body of the request. The body should be hashed using SHA-1 and encoded using Base64. All h breaks every 60 characters.
`X-Ops-Server-API-Version`	Use `X-Ops-Server-API-Version` to specify the version of the Chef Infra Server API. For example: `X-` Chef Server version 12, but will be deprecated as part of the next major release.
`X-Ops-Sign`	Set this header to the following value: `version=1.0`.
`X-Ops-Timestamp`	The timestamp, in ISO-8601 format and with UTC indicated by a trailing `Z` and separated by the cha
`X-Ops-UserId`	The name of the API client whose private key will be used to create the authorization header.

{{< note >}}

Use `X-Ops-Server-API-Info` to identify the version of the Chef Infra Server API.

{{< /note >}}

Example

The following example shows an authentication request:

```
GET /organizations/NAME/nodes HTTP/1.1
  Accept: application/json
  Accept-Encoding: gzip;q=1.0,deflate;q=0.6,identity;q=0.3
  X-Ops-Sign: algorithm=sha1;version=1.0;
  X-Ops-Userid: user_id
  X-Ops-Timestamp: 2014-12-12T17:13:28Z
  X-Ops-Content-Hash: 2jmj7l5rfasfgSw0ygaVb/vlWAghYkK/YBwk=
  X-Ops-Authorization-1: BE3NnBritishaf3ifuwLSPCCYasdfXaRN5oZb4c6hbW0aefI
  X-Ops-Authorization-2: sL4j1qtEZzi/2WeF67UuytdsdfgbOc5CjgECQwqrym9gCUON
  X-Ops-Authorization-3: yf0p7PrLRCNasdfaHhQ2LWSea+kTcu0dkasdfvaTghfCDC57
  X-Ops-Authorization-4: 155i+ZlthfasfasdffukusbIUGBKUYFjhbvcds3k0i0gqs+V
  X-Ops-Authorization-5: /sLcR7JjQky7sdafIHNfsBQrISktNPower1236hbFIayFBx3
  X-Ops-Authorization-6: nodilAGMb166@haC/fttwlWQ2N1LasdqqGomRedtyhSqXA==
  Host: api.opscode.com:443
  X-Ops-Server-API-Info: 1
  X-Chef-Version: 12.0.2
  User-Agent: Chef Knife/12.0.2 (ruby-2.1.1-p320; ohai-8.0.0; x86_64-darwin12.0.2; +http://chef.io)
```

Knife API Requests

{{% plugin_knife_summary %}}

{{% plugin_knife_using_authenticated_requests %}}

Global Endpoints

A global endpoint may be used to access all of the organizations on the Chef Infra Server.

/authenticate_user

The `/authenticate_user` endpoint has the following methods: `POST`.

POST

The `POST` method is used to authenticate a user. This endpoint is used by the Chef Identity Service to authenticate users of Chef Supermarket to the Chef server.

This method has no parameters.

Request

```
POST /authenticate_user
```

with a request body similar to:

```
{
  "username" : "grantmc",
  "password" : "p@ssw0rd"
}
```

Response

This method has no response body.

Response Codes

Response Code	Description
`200`	OK. The request was successful.
`401`	Unauthorized. The user or client who made the request could not be authenticated. Verify the user/cli request.

/license

{{< note >}}

This endpoint is used for information purposes only and to trigger a notification in the Chef management console about the number of licenses owned vs. the nu behavior of the Chef Infra Server and any added component does not change.

{{< /note >}}

The `/license` endpoint has the following methods: `GET`.

GET

The `GET` method is used to get license information for the Chef Infra Server.

This method has no parameters.

Request

```
GET /license
```

This method has no request body.

Response

The response is similar to:

```
{
  "limit_exceeded": false,
  "node_license": 25,
  "node_count": 12,
  "upgrade_url": "http://www.chef.io/contact/on-premises-simple"
}
```

When `node_count` is greater than `node_license`, then `limit_exceeded` is `true` and the Chef management console will display a notification about this status. The then update the configuration settings appropriately.

The chef-server.rb file contains settings that can be used to edit the number of nodes that are under license:

Setting	Description

Setting	Description
`license['nodes']`	The number of licensed nodes. Default value: `25`.
`license['upgrade_url']`	The URL to visit for more information about how to update the number of nodes licensed for an organi: `simple"`.

Response Codes

Response Code	Description
`200`	OK. The request was successful.
`401`	Unauthorized. The user or client who made the request could not be authenticated. Verify the use
`403`	Forbidden. The user who made the request is not authorized to perform the action.

/organizations

The Chef Infra Server may contain multiple organizations.

The `/organizations` endpoint has the following methods: `GET` and `POST`.

{{< warning >}}

This endpoint may only be accessed by the `pivotal` user, which is created as part of the installation process for the Chef Infra Server. (See the "Query for Users ; user.)

{{< /warning >}}

GET

The `GET` method is used to get a list of organizations on the Chef Infra Server.

Request

```
GET /organizations
```

Response

The response is similar to:

```
{
  "org_name1": https://url/for/org_name1",
  "org_name2": https://url/for/org_name2"
}
```

Response Codes

Response Code	Description
`200`	OK. The request was successful.
`403`	Forbidden. The user who made the request is not authorized to pe

POST

The `POST` method is used to create an organization on the Chef Infra Server.

This method has no parameters.

Request

```
POST /organizations
```

with a request body similar to:

```
{
  "name": "org_name1",
  "full_name": "Org_name1 Full Name"
}
```

where:

- `name` must begin with a lower-case letter or digit, may only contain lower-case letters, digits, hyphens, and underscores, and must be between 1 and 255 chara
- `full_name` must begin with a non-white space character and must be between 1 and 1023 characters. For example: `Chef Software, Inc.`.

An organization isn't usable until a user that belongs to the `admins` group is associated with the organization.

{{< /note >}}

Response

The response is similar to:

```
{
  "clientname": "org_name1-validator",
  "private_key": "-----BEGIN RSA PRIVATE KEY----- MIIEpQIBAAKCAQEAx2uyX ...",
  "uri": "https://url/for/org_name1"
}
```

Response Codes

Response Code	Description
`201`	Created. The request was successful. The organization was create
`400`	Bad request. The contents of the request are not formatted correct
`403`	Forbidden. The user who made the request is not authorized to pe
`409`	Conflict. The organization already exists.

/organizations/NAME

An organization is a single instance of a Chef Infra Server, including all of the nodes that are managed by that Chef Infra Server and each of the workstations that wil

The `/organizations/NAME` endpoint has the following methods: `DELETE`, `GET`, and `PUT`.

DELETE

The `DELETE` method is used to delete an organization.

This method has no parameters.

Request

```
DELETE /organizations/NAME
```

Response

The response is similar to:

```
{
  "name": "chef",
  "full_name": "Chef Software, Inc",
  "guid": "f980d1asdfda0331235s00ff36862"
}
```

Response Codes

Response Code	Description

Response Code	Description
`200`	OK. The request was successful.
`403`	Forbidden. The user who made the request is not authorized to pe

GET

The `GET` method is used to get the details for the named organization.

This method has no parameters.

Request

```
GET /organizations/NAME
```

Response

The response is similar to:

```
{
  "name": "chef",
  "full_name": "Chef Software, Inc",
  "guid": "f980d1asdfda0331235s00ff36862"
     ...
}
```

Response Codes

Response Code	Description
`200`	OK. The request was successful.
`403`	Forbidden. The user who made the request is not authorized to pe

PUT

The `PUT` method is used to update an organization definition.

This method has no parameters.

Request

```
PUT /organizations/NAME
```

with a request body similar to:

```
{
  "name": "chef",
  "full_name": "Chef Software, Inc"
}
```

Response

The response is similar to:

```
{
  "name": "chef",
  "full_name": "Chef Software, Inc",
  "guid": "f980d1asdfda0331235s00ff36862"
}
```

Response Codes

Response Code	Description
`200`	OK. The request was successful.
`400`	Bad request. The contents of the request are not formatted correct
`403`	Forbidden. The user who made the request is not authorized to pe
`410`	Gone. Unable to update private key.

/_status

Use the `/_status` endpoint to check the status of communications between the front and back end servers. This endpoint is located at `/_status` on the front end s

GET

The `GET` method is used to get the details for the named organization.

Request

```
GET /_status
```

This method has no parameters. This method has no request body.

Response

The response will return something like the following:

```
{
  "status": "pong",
  "upstreams":
    {
      "service_name": "pong",
      "service_name": "pong",
      ...
    }
  "keygen":
    {
      "keys": 10,
      ....
    }
 }
```

Response Codes

Response Code	Description
`200`	All communications are OK.

`500`	One (or more) services are down. For example:
	`{`
	`  "status":"fail",`
	`  "upstreams":`
	`    {`
	`      "service_name": "fail",`
	`      "service_name": "pong",`
	`      ...`
	`    }`
	`}`

/users

A user is an individual account that is created to allow access to the Chef Infra Server. For example:

- A hosted Chef Infra Server account
- The user that operates the workstation from which a Chef Infra Server will be managed

The `/users` endpoint has the following methods: `GET` and `POST`.

{{< warning >}}

This endpoint may only be accessed by the `pivotal` user, which is created as part of the installation process for the Chef Infra Server. (See the "Query for Users
user.)

{{< /warning >}}

{{< note >}}

This documentation for the `/users` endpoint is for version 1 of the Chef Infra Server API. Version 0 of the API has some differences in the request body and in the re

{{< /note >}}

GET

The `GET` method is used to get a list of users on the Chef Infra Server.

This method has the following parameters:

Parameter	Description
`email=jane@chef.com`	Filter the users returned based on their email id.
`external_authentication_uid=jane@chef.com`	Filter the users returned based on their external login id.
`verbose=true`	Returns a user list with "email", "first_name", "last_name" f ignored.

Request

```
GET /users
```

Response

The response is similar to:

```
{
  "user1": "https://chef.example/users/user1",
  "user2": "https://chef.example/users/user2"
}
```

The verbose response is similar to:

```
{
  "janechef": { "email": "jane.chef@user.com", "first_name": "jane", "last_name": "chef_user" },
  "yaelsmith": { "email": "yeal.chef@user.com", "first_name": "yeal", "last_name": "smith" }
}
```

Response Codes

Response Code	Description
`200`	OK. The request was successful.
`401`	Unauthorized. The user or client who made the request could not be authenticated. Verify the use
`403`	Forbidden. The user who made the request is not authorized to perform the action.
`404`	Not found. The requested object does not exist.

Optional Filtering

Filtering on `/users` can be done with the `external_authentication_uid`. This is to support SAML authentication.

As an example, to retrieve users whos `external_authentication_uid` is `jane@doe.com`, you would do the following:

```
GET /users?external_authentication_uid=jane%40doe.com
```

New in Chef Server 12.7.

POST

The `POST` method is used to create a user on the Chef Infra Server.

This method has no parameters.

Request

```
POST /users
```

with a request body similar to:

```
{
  "username": "robert-forster",
  "display_name": "robert",
  "email": "robert@noreply.com",
  "first_name": "robert",
  "last_name": "forster",
  "middle_name": "",
  "password": "yeahpass",
  "create_key": true,
  "public_key": "-----BEGIN PUBLIC KEY-----\nMIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAoYyN0AIhUh7Fw1+gQtR+ \n0/HY362
majqKEnNywN8/NByZhhlLdBxBX/UN04/7aHZMoZxrrjXGLcyjvXN3uxyCO\nyPY989pa68LJ9jXWyyfKjCYdztSFcRuwF7tWgqnlsc8pve/UaWamNOTXQny
Hl9U196\n06Ajv1RNnfyHnBXIM+I5mxJRyJCyDFo/MACc5AgO6M0a7sJ/sdX+WccgcHEVbPAl\n1wIDAQAB \n-----END PUBLIC KEY-----\n\n"
}
```

where:

- `username` must begin with a lower-case letter or digit, may only contain lower-case letters, digits, hyphens, and underscores. For example: `chef`. `username` is r
  elements matching `a-z0-9!#$%&'*+/=?^_`{|}~-`.
- `display_name` is required to be present.
- `email` is required to be present and have a valid value. The email validation doesn't allow for all unicode characters.
- Either `external_authentication_uid` or `password` are required to be present and have a value.
- During the POST, the `public_key` value will be broken out and resubmitted to the keys portion of the API in the latest Chef Infra Server versions.
- Only one of the keys, `create_key` or `public_key`, may be specified. If `create_key` is specified, a default private key is generated and returned.

Response

The response is similar to:

```
{
  "uri": "https://chef.example/users/robert-forster",
  "chef_key": {
    "name": "default",
    "public_key": "-----BEGIN RSA PUBLIC KEY...",
    "expiration_date": "infinity",
    "uri": "https://chef.example/users/robert-forster/keys/default",
    "private_key": "-----BEGIN RSA PRIVATE KEY..."
  }
}
```

Response Codes

Response Code	Description
`201`	OK. The user was created.

Response Code	Description
`400`	Bad request. The contents of the request are not formatted correctly.
`401`	Unauthorized. The user or client who made the request could not be authenticated. Verify the use
`403`	Forbidden. The user who made the request is not authorized to perform the action.
`409`	Conflict. The object already exists.
`413`	Request entity too large. A request may not be larger than 1000000 bytes.

/users/NAME

The `/users/USER_NAME` endpoint has the following methods: `DELETE`, `GET`, and `PUT`.

{{< note >}}

This documentation for the `/users/NAME` endpoint is for version 1 of the Chef Infra Server API. Version 0 of the API has some differences in the request body and in

{{< /note >}}

DELETE

The `DELETE` method is used to delete a user.

This method has no parameters.

Request

```
DELETE /users/USER_NAME
```

Response

The response is similar to:

```
{
}
```

Response Codes

Response Code	Description
`200`	OK. The request was successful.
`401`	Unauthorized. The user or client who made the request could not be authenticated. Verify the use
`403`	Forbidden. The user who made the request is not authorized to perform the action.
`404`	Not found. The requested object does not exist.

GET

The `GET` method is used to return the details for a user.

This method has no parameters.

Request

```
GET /users/USER_NAME
```

Response

The response is similar to:

```
{
  "username": "robert-forster",
  "display_name": "robert",
  "email": "robert@noreply.com",
  "first_name": "robert",
  "last_name": "forster"
}
```

Response Codes

Response Code	Description
`200`	OK. The request was successful.
`401`	Unauthorized. The user or client who made the request could not be authenticated. Verify the use
`403`	Forbidden. The user who made the request is not authorized to perform the action.
`404`	Not found. The requested object does not exist.

PUT

The `PUT` method is used to update a specific user. If values are not specified for the `PUT` method, the Chef Infra Server will use the existing values rather than assign

{{< note >}}

`PUT` supports renames. If `PUT /users/foo` is requested with `{ "username: "bar""}`, then it will rename `foo` to `bar` and all of the content previously associated wit

{{< /note >}}

{{< note >}}

As of 12.1.0, the `"public_key"`, `"private_key"`, and `"create_key"` parameters in PUT requests to clients/users will cause a 400 response.

{{< /note >}}

This method has no parameters.

Request

```
PUT /users/NAME
```

with a request body similar to:

```
{
  "username":     "grant.mclennan",
  "display_name": "Grant McLennan",
  "email":        "grant@newlocation.com",
  "first_name":   "Grant",
  "last_name":    "McLennan",
  "middle_name":  "james",
  "public_key" : "-------- BEGIN PUBLIC KEY ----and a valid key here"
}
```

Response

The response is similar to:

```
{
  "uri": "https://chef.example/users/grant.mclennan",
  "chef_key": {
    "name": "default",
    "public_key": "-----BEGIN RSA PUBLIC KEY...",
    "expiration_date": "infinity",
    "uri": "https://chef.example/users/rober-forster/keys/default",
    "private_key": ""
  }
}
```

If a new private key was generated, both the private and public keys are returned.

Response Codes

Response Code	Description
`200`	OK. The request was successful.
`201`	Created. The object was created. (This response code is only returned when the user is renamed.)
`400`	Invalid. Invalid or missing values. Otherwise malformed request.
`401`	Unauthorized. The user or client who made the request could not be authenticated. Verify the use
`403`	Forbidden. The user who made the request is not authorized to perform the action.
`404`	Not found. The requested object does not exist.
`409`	Conflict. This response code is only returned when a user is renamed, but a user already exists wi
`413`	Request entity too large. A request may not be larger than 1000000 bytes.

/users/USER/keys/

The `/users/USER/keys` endpoint has the following methods: `GET` and `POST`. User keys are public RSA keys in the SSL `.pem` file format and are used for authenticati

GET

The `GET` method is used to retrieve all of the named user's key identifiers, associated URIs, and expiry states.

This method has no parameters.

Request

```
GET /users/USER/keys/
```

Response

The response is similar to:

```
[
  {
    "name" : "default",
    "uri" : "https://chef.example/users/USER/keys/default",
    "expired" : false
  },
  {
    "name" : "key1",
    "uri" : "https://chef.example/users/USER/keys/key1",
    "expired" : false
  }
]
```

Response Codes

Response Code	Description
`200`	OK. The request was successful.
`401`	Unauthorized. The user or client who made the request could not be authenticated. Verify the use
`403`	Forbidden. The user who made the request is not authorized to perform the action.
`404`	Not found. The requested object does not exist.

POST

The `POST` method is used to add a key for the specified user.

This method has no parameters.

Request

```
POST /users/USER/keys/
```

with a request body similar to:

```
{
  "name" : "key1",
  "public_key" : "-------- BEGIN PUBLIC KEY ----and a valid key here",
  "expiration_date" : "infinity"
}
```

Response

The response is similar to:

```
{
  "name" : "key1",
  "uri" : "https://chapi_chef_server.mdef.example/users/user1/keys/key1",
  "expired": false
}
```

Response Codes

Response Code	Description
`201`	Created. The object was created.
`401`	Unauthorized. The user or client who made the request could not be authenticated. Verify the use
`403`	Forbidden. The user who made the request is not authorized to perform the action.
`404`	Not found. The requested object does not exist.

/users/USER/keys/KEY

The `/users/USER/keys/KEY` endpoint has the following methods: `DELETE`, `GET`, and `PUT`.

DELETE

The `DELETE` method is used to delete the specified key for the specified user.

This method has no parameters.

Request

```
DELETE /users/USER/keys/KEY
```

Response

The response returns the information about the deleted key and is similar to:

```
{
  "name" : "default",
  "public_key" : "-------- BEGIN PUBLIC KEY --------- ...",
  "expiration_date" : "2020-12-31T00:00:00Z"
}
```

Response Codes

Response Code	Description

Response Code	Description
`200`	OK. The request was successful.
`401`	Unauthorized. The user or client who made the request could not be authenticated. Verify the use
`403`	Forbidden. The user who made the request is not authorized to perform the action.
`404`	Not found. The requested object does not exist.

GET

The `GET` method is used to return details for a specific key for a specific user.

This method has no parameters.

Request

```
GET /users/USER/keys/KEY
```

Response

The response is similar to:

```
{
  "name" : "default",
  "public_key" : "-------- BEGIN PUBLIC KEY --------- ...",
  "expiration_date" : "2020-12-31T00:00:00Z"
}
```

Response Codes

Response Code	Description
`200`	OK. The request was successful.
`401`	Unauthorized. The user or client who made the request could not be authenticated. Verify the use
`403`	Forbidden. The user who made the request is not authorized to perform the action.
`404`	Not found. The requested object does not exist.

PUT

The `PUT` method is used to update one or more properties for a specific key for a specific user.

This method has no parameters.

Request

```
PUT /users/USER/keys/KEY
```

with a request body similar to:

```
{
  "name" : "new_key_name",
  "public_key" : "-------- BEGIN PUBLIC KEY ----and a valid key here",
  "expiration_date" : "2020-12-31T00:00:00Z"
}
```

Response

The response contains the updated inforamtion for the key, and is similar to:

```
{
  "name" : "new_key_name",
  "public_key" : "-------- BEGIN PUBLIC KEY --------- ...",
  "expiration_date" : "2020-12-31T00:00:00Z"
}
```

Response Codes

Response Code	Description
`200`	OK. The request was successful.
`201`	Created. The object was created.
`401`	Unauthorized. The user or client who made the request could not be authenticated. Verify the use
`403`	Forbidden. The user who made the request is not authorized to perform the action.
`404`	Not found. The requested object does not exist.

Organization Endpoints

Each organization-specific authentication request must include `/organizations/NAME` as part of the name for the endpoint. For example, the full endpoint for getting

```
GET /organizations/NAME/roles
```

where `ORG_NAME` is the name of the organization.

/association_requests

Users may be invited to join organizations via the web user interface in the Chef management console or via the `POST` endpoint in the Chef Infra Server API.

The `/association_requests` endpoint has the following methods: `DELETE`, `GET`, and `POST`.

DELETE

The `DELETE` method is used to delete a pending invitation.

This method has no parameters.

Request

```
DELETE /organizations/NAME/association_requests/ID
```

This method has no request body.

Response

The response is similar to:

```
{
  "id":      "79b9382ab70e962907cee1747f9969a4",
  "orgname": "testorg",
  "username" "janedoe"
}
```

Response Codes

Response Code	Description
`200`	OK. The request was successful.

Response Code	Description
`401`	Unauthorized. The user or client who made the request could not be authenticated. Verify the use
`403`	Forbidden. The user who made the request is not authorized to perform the action.
`404`	Not found. The requested object does not exist.

GET

The `GET` method is used to get a list of pending invitations.

This method has no parameters.

Request

```
GET /organizations/NAME/association_requests
```

This method has no request body.

Response

The response returns a dictionary similar to:

```
[
  {
    "id": "79b9382ab70e962907cee1747f9969a4",
    "username": "marygupta"
  },
  {
    "id": "24t1432uf33x799382abb7096g8190b5",
    "username": "johnirving"
  }
]
```

Response Codes

Response Code	Description
`200`	OK. The request was successful.
`401`	Unauthorized. The user or client who made the request could not be authenticated. Verify the use
`403`	Forbidden. The user who made the request is not authorized to perform the action.

POST

The `POST` method is used to create an invitation.

This method has no parameters.

Request

```
{
 "user": "billysmith"
}

POST /organizations/NAME/association_requests
```

Response

The response is similar to:

```
{
  "uri": "https://chef.example/organizations/test/association_requests/79b9382ab70e962907cee1747f9969a4",
  "organization_user": {
    "username": "authorizeduser"
  },
  "organization": {
    "name": "test"
  },
  "user": {
    "email": "sallyjane@domain.org",
    "first_name": "sally"
  }
}
```

Response Codes

Response Code	Description
`201`	OK. An invitation was created.
`400`	Bad request. The contents of the request are not formatted correctly.
`401`	Unauthorized. The user or client who made the request could not be authenticated. Verify the use
`403`	Forbidden. The user who made the request is not authorized to perform the action.
`404`	Not found. The invited user does not exist.
`409`	Conflict. The object already exists.

/clients

Use the `/clients` endpoint to manage clients and their associated RSA key-pairs. The `/clients` endpoint has the following methods: `GET` and `POST`.

{{< note >}}

The clients should be managed using knife as opposed to the Chef Infra Server API. The interactions between clients, nodes and acls are tricky.

{{< /note >}}

GET

The `GET` method is used to return a client list on the Chef Infra Server, including clients for nodes that have been registered with the Chef Infra Server, the chef-valida

This method has no parameters.

Request

```
GET /organizations/NAME/clients
```

This method has no request body.

Response

The response is similar to:

```
{
  "org1-validator" : "https://chef.example/orgaizations/org1/clients/org1-validator",
  "client1" : "https://chef.example/orgaizations/org1/clients/client1"
}
```

Response Codes

Response Code	Description
`200`	OK. The request was successful.
`401`	Unauthorized. The user or client who made the request could not be authenticated. Verify the use
`403`	Forbidden. The user who made the request is not authorized to perform the action.

POST

The `POST` method is used to create a new API client.

{{< note >}}

As of 12.1.0, the `"admin"` parameter is no longer supported in client/user creation and support. If used in the `POST` or `PUT` of a client or user, the `"admin"` paramete

{{< /note >}}

This method has no parameters.

Request

```
POST /organizations/NAME/clients
```

with a request body similar to:

```
{
  "name": "name_of_API_client",
  "clientname": "name_of_API_client",
  "validator": true,
  "create_key": true
}
```

where `name_of_API_client` is the name of the API client to be created and `admin` indicates whether the API client will be run as an admin API client. Either name or

Response

The response is similar to:

```
{
  "uri": "https://chef.example/orgaizations/org1/clients/client1",
  "chef_key": {
    "name": "default",
    "expiration_date": "infinity",
    "private_key": "-----BEGIN RSA PRIVATE KEY----- ...",
    "public_key": "-----BEGIN PUBLIC KEY----- ... ",
    "uri": "https://chef.example/orgaizations/org1/clients/client1/keys/default"
}
```

Store the private key in a safe place. It will be required later (along with the client name) to access the Chef Infra Server when using the Chef Infra Server API.

Response Codes

Response Code	Description
`201`	Created. The client was created.
`400`	Bad request. The contents of the request are not formatted correctly.
`401`	Unauthorized. The user or client who made the request could not be authenticated. Verify the use
`403`	Forbidden. The user who made the request is not authorized to perform the action.
`409`	Conflict. The object already exists.
`413`	Request entity too large. A request may not be larger than 1000000 bytes.

/clients/NAME

The `/clients/NAME` endpoint is used to manage a specific client. This endpoint has the following methods: `DELETE`, `GET`, and `PUT`.

DELETE

The `DELETE` method is used to remove a specific client.

This method has no parameters.

Request

```
DELETE /organizations/NAME/clients/NAME
```

This method has no request body.

Response

The response has no body.

Response Codes

Response Code	Description
`200`	OK. The request was successful.
`401`	Unauthorized. The user or client who made the request could not be authenticated. Verify the use
`403`	Forbidden. The user who made the request is not authorized to perform the action.
`404`	Not found. The requested object does not exist.

GET

The `GET` method is used to return a specific API client.

This method has no parameters.

Request

```
GET /organizations/NAME/clients/NAME
```

This method has no request body.

Response

The response is similar to:

```
{
  "name": "user1",
  "clientname": "user1",
  "orgname": "test",
  "json_class": "Chef::ApiClient",
  "chef_type": "client",
  "validator": "false"
}
```

Response Codes

Response Code	Description
`200`	OK. The request was successful.
`401`	Unauthorized. The user or client who made the request could not be authenticated. Verify the use
`403`	Forbidden. The user who made the request is not authorized to perform the action.
`404`	Not found. The requested object does not exist.

PUT

The `PUT` method is used to update a specific client. If values are not specified for the `PUT` method, the Chef Infra Server will use the existing values rather than assig

{{< note >}}

`PUT` supports renames. If `PUT /client/foo` is requested with `{ "name: "bar""}`, then it will rename `foo` to `bar` and all of the content previously associated with `fo

{{< /note >}}

{{< note >}}

As of 12.1.0, the `"admin"` parameter is no longer supported in client/user creation and support. If used in the `POST` or `PUT` of a client or user, then it is ignored.

{{< /note >}}

{{< note >}}

As of 12.1.0, including `"public_key"`, `"private_key"`, or `"create_key"` in PUT requests to clients/users will cause a 400 response.

{{< /note >}}

{{< note >}}

`"name"` and `"clientname"` are not independent values. Making a PUT request with different values will return a 400 error. Either name may be specified to set both

{{< /note >}}

Request

```
PUT /organizations/NAME/clients/NAME
```

with a request body similar to:

```
{
  "name": "monkeypants",
  "validator": false
}
```

Response

The response is similar to:

```
{
  "name": "monkeypants",
  "clientname": "monkeypants",
  "validator": true,
  "json_class":"Chef::ApiClient",
  "chef_type":"client"
}
```

Response Codes

Response Code	Description
`200`	OK. The request was successful.
`201`	Created. The client was updated. (This response code is only returned when the client is renamed
`401`	Unauthorized. The user or client who made the request could not be authenticated. Verify the use
`403`	Forbidden. The user who made the request is not authorized to perform the action.
`404`	Not found. The requested object does not exist.
`409`	Conflict. This response code is only returned when a client is renamed, but a client already exists
`413`	Request entity too large. A request may not be larger than 1000000 bytes.

/clients/CLIENT/keys/

The `/clients/CLIENT/keys` endpoint has the following methods: `GET` and `POST`.

GET

The `GET` method is used to retrieve all of the named client's key identifiers, associated URIs, and expiry states.

This method has no parameters.

Request

```
GET /organizations/NAME/clients/CLIENT/keys
```

This method has no request body.

Response

The response is similar to:

```
[
  {
     "name": "default",
     "uri": "https://chef.example/organizations/example/clients/client1/keys/default",
     "expired": false
  },
  {
     "name": "key1",
     "uri": "https://chef.example/organizations/example/clients/client1/keys/key1",
     "expired": true
  }
]
```

Response Codes

Response Code	Description
`200`	OK. The request was successful.
`401`	Unauthorized. The user or client who made the request could not be authenticated. Verify the use
`403`	Forbidden. The user who made the request is not authorized to perform the action.
`404`	Not found. The requested object does not exist.

POST

The `POST` method is used to add a key for the specified client.

This method has no parameters.

Request

```
POST /organizations/NAME/clients/CLIENT/keys
```

with a request body similar to:

```
{
  "name": "key1",
  "public_key": "-------- BEGIN PUBLIC KEY ----and a valid key here",
  "expiration_date": "infinity"
}
```

Response

The response is similar to:

```
{
  "uri": "https://chef.example/organizations/example/clients/client1/keys/key1"
}
```

Response Codes

Response Code	Description
`201`	Created. The object was created.
`401`	Unauthorized. The user or client who made the request could not be authenticated. Verify the use

Response Code	Description
`403`	Forbidden. The user who made the request is not authorized to perform the action.
`404`	Not found. The requested object does not exist.

/clients/CLIENT/keys/KEY

The `/clients/CLIENT/keys/KEY` endpoint has the following methods: `DELETE`, `GET`, and `PUT`.

DELETE

The `DELETE` method is used to delete the specified key for the specified client.

This method has no parameters.

Request

```
DELETE /organizations/NAME/clients/CLIENT/keys/KEY
```

This method has no request body.

Response

The response returns the information about the deleted key and is similar to:

```
{
  "name": "default",
  "public_key": "-------- BEGIN PUBLIC KEY --------- ...",
  "expiration_date": "2020-12-31T00:00:00Z"
}
```

Response Codes

Response Code	Description
`200`	OK. The request was successful.
`401`	Unauthorized. The user or client who made the request could not be authenticated. Verify the use
`403`	Forbidden. The user who made the request is not authorized to perform the action.
`404`	Not found. The requested object does not exist.

GET

The `GET` method is used to return details for a specific key for a specific client.

This method has no parameters.

Request

```
GET /organizations/NAME/clients/CLIENT/keys/KEY
```

This method has no request body.

Response

The response is similar to:

```
{
  "name" : "default",
  "public_key" : "-------- BEGIN PUBLIC KEY --------- ...",
  "expiration_date" : "2020-12-31T00:00:00Z"
}
```

Response Codes

Response Code	Description
`200`	OK. The request was successful.
`401`	Unauthorized. The user or client who made the request could not be authenticated. Verify the use
`403`	Forbidden. The user who made the request is not authorized to perform the action.
`404`	Not found. The requested object does not exist.

PUT

The `PUT` method is used to update one or more properties for a specific key for a specific client.

This method has no parameters.

Request

```
PUT /organizations/NAME/clients/CLIENT/keys/KEY
```

with a request body similar to:

```
{
  "name": "new_key_name",
  "public_key": "-------- BEGIN PUBLIC KEY ----and a valid key here",
  "expiration_date": "2020-12-31T00:00:00Z"
}
```

Response

The response contains the updated information for the key and is similar to:

```
{
  "name": "new_key_name",
  "public_key": "-------- BEGIN PUBLIC KEY --------- ...",
  "expiration_date": "2020-12-31T00:00:00Z"
}
```

Response Codes

Response Code	Description
`200`	OK. The request was successful.
`201`	Created. The object was created.
`401`	Unauthorized. The user or client who made the request could not be authenticated. Verify the use
`403`	Forbidden. The user who made the request is not authorized to perform the action.
`404`	Not found. The requested object does not exist.

/containers -----------

The `/containers` endpoint has the following methods: `GET`.

GET

The `GET` method is used to get a list of containers.

This method has no parameters.

Request

```
GET /organizations/NAME/containers
```

Response

The response is similar to:

```
{
  "clients": "https://url/for/containers/clients",
  "containers": "https://url/for/containers/containers",
  "cookbooks": "https://url/for/containers/cookbooks",
  "data": "https://url/for/containers/data",
  "environments": "https://url/for/containers/environments",
  "groups": "https://url/for/containers/groups",
  "nodes": "https://url/for/containers/nodes",
  "roles": "https://url/for/containers/roles",
  "sandboxes": "https://url/for/containers/sandboxes"
}
```

Response Codes

Response Code	Description
`200`	OK. The request was successful.
`401`	Unauthorized. The user or client who made the request could not be authenticated. Verify the use
`403`	Forbidden. The user who made the request is not authorized to perform the action.
`404`	Not found. The requested object does not exist.

/cookbook_artifacts

Cookbook artifacts are specific versions of cookbooks that were specified by a Policyfile applied to a node.

The `/organization/NAME/cookbook_artifacts` endpoint has the following methods: `GET`.

GET

The `GET` method is used to return a hash of all cookbook artifacts and their versions.

This method has no parameters.

Request

```
GET /organizations/NAME/cookbook_artifacts
```

This method has no request body.

Response

The response is similar to:

```
{
  "oc-influxdb": {
    "url": "https://chef.example/organizations/example-org/cookbook_artifacts/oc-influxdb",
    "versions": [
      {
        "url": "https://chef.example/organizations/example-org/cookbook_artifacts/oc-influxdb/9634a5d998b02ff069761f6e13
        "identifier": "9634a5d998b02ff069761f6e1309a41572d0f858"
      },
      {
        "url": "https://chef.example/organizations/example-org/cookbook_artifacts/oc-influxdb/d774c9bb079f21b64c34275ecd
        "identifier": "d774c9bb079f21b64c34275ecd4b371e0cae71a1"
      }
    ]
  },
  "rabbitmq": {
    "url": "https://chef.example/organizations/example-org/cookbook_artifacts/rabbitmq",
    "versions": [
      {
        "url": "https://chef.example/organizations/example-org/cookbook_artifacts/rabbitmq/58035a5b41c005f3b5b98f22ccae
        "identifier": "58035a5b41c005f3b5b98f22ccaed1a0d6161e22"
      },
      {
        "url": "https://chef.example/organizations/example-org/cookbook_artifacts/rabbitmq/5c08f92cc01f94ee37d382c32023b
        "identifier": "5c08f92cc01f94ee37d382c32023b137ee343a1e"
      }
    ]
  }
}
```

Response Codes

Response Code	Description
`200`	OK. The request was successful.
`401`	Unauthorized. The user or client who made the request could not be authenticated. Verify the use
`403`	Forbidden. The user who made the request is not authorized to perform the action.

/cookbook_artifacts/NAME

This endpoint lists versions of a named cookbook artifact.

The `/organization/NAME/cookbook_artifacts/NAME` endpoint has the following methods: `GET`.

GET

The `GET` method is used to return a hash of a single cookbook artifact and its versions.

This method has no parameters.

Request

```
GET /organizations/NAME/cookbook_artifacts/NAME
```

This method has no request body.

Response

The response is similar to:

```
{
  "rabbitmq": {
    "url": "https://chef.example/organizations/example-org/cookbook_artifacts/rabbitmq",
    "versions": [
      {
        "url": "https://chef.example/organizations/example-org/cookbook_artifacts/rabbitmq/0bd7539be0434e3355aff8ecccf45
        "identifier": "0bd7539be0434e3355aff8ecccf4543ecf5c4be2"
      },
      {
        "url": "https://chef.example/organizations/example-org/cookbook_artifacts/rabbitmq/0e1016d364685b87456c648136da0
        "identifier": "0e1016d364685b87456c648136da04a2559821ec"
      }
    ]
  }
}
```

Response Codes

Response Code	Description
`200`	OK. The request was successful.
`401`	Unauthorized. The user or client who made the request could not be authenticated. Verify the use
`403`	Forbidden. The user who made the request is not authorized to perform the action.

Response Code	Description
`404`	Not found. The requested object does not exist.

/cookbook_artifacts/NAME/ID

The `/organization/NAME/cookbook_artifacts/NAME/ID` endpoint has the following methods: `DELETE`, `GET`, and `PUT`.

DELETE

The `DELETE` method is used to delete a single cookbook artifact version.

This method has no parameters.

Request

```
DELETE /organizations/NAME/cookbook_artifacts/NAME/ID
```

This method has no request body.

Response

The response contains the record of the deleted resource and is similar to:

```
{
  "version": "5.7.7",
  "name": "rabbitmq",
  "identifier": "f3cf8ea7d8bfc59e35ec541946e3e82cd4b73e74",
  "frozen?": false,
  "chef_type": "cookbook_version",
  "attributes": [
    {
      "name": "default.rb",
      "path": "attributes/default.rb",
      "checksum": "e5a530cca3898d8bd07604435dc5156e",
      "specificity": "default",
      "url": "https://chef.example/bookshelf/organization-9f69768696feedcd165633b8b475cc0b/checksum-e5a530cca3898d8bd07
    }
  ],
  "definitions": [
  ],
  "files": [
  ],
  "libraries": [
    {
      "name": "matchers.rb",
      "path": "libraries/matchers.rb",
      "checksum": "24c3f44c4d1d62300a56051f0069f639",
      "specificity": "default",
      "url": "https://chef.example/bookshelf/organization-9f69768696feedcd165633b8b475cc0b/checksum-24c3f44c4d1d62300a56
    },
    {
      "name": "helpers.rb",
      "path": "libraries/helpers.rb",
      "checksum": "df65c4a7259fcb30c6f3f1305ebf7502",
      "specificity": "default",
      "url": "https://chef.example/bookshelf/organization-9f69768696feedcd165633b8b475cc0b/checksum-df65c4a7259fcb30c6f3
    },
    {
      "name": "default.rb",
      "path": "libraries/default.rb",
      "checksum": "94292faac84ba797e720501700b30f74",
      "specificity": "default",
      "url": "https://chef.example/bookshelf/organization-9f69768696feedcd165633b8b475cc0b/checksum-94292faac84ba797e720
    }
  ],
  "providers": [
    {
      "name": "user.rb",
      "path": "providers/user.rb",
      "checksum": "c31c9cc749f21962c825f983a6679d94",
      "specificity": "default",
      "url": "https://chef.example/bookshelf/organization-9f69768696feedcd165633b8b475cc0b/checksum-c31c9cc749f21962c825
    },
    {
      "name": "policy.rb",
      "path": "providers/policy.rb",
      "checksum": "746c8a3f248f5bbfa51f5d2ba60b6315",
      "specificity": "default",
      "url": "https://chef.example/bookshelf/organization-9f69768696feedcd165633b8b475cc0b/checksum-746c8a3f248f5bbfa51
    }
  ],
  "recipes": [
    {
      "name": "default.rb",
      "path": "recipes/default.rb",
      "checksum": "99a9b404ff6038d6ac55a90ca68c347a",
      "specificity": "default",
      "url": "https://chef.example/bookshelf/organization-9f69768696feedcd165633b8b475cc0b/checksum-99a9b404ff6038d6ac5
    },
    {
      "name": "cluster.rb",
      "path": "recipes/cluster.rb",
      "checksum": "fc0a86c1f858c9d37e11282efc9fe329",
      "specificity": "default",
      "url": "https://chef.example/bookshelf/organization-9f69768696feedcd165633b8b475cc0b/checksum-fc0a86c1f858c9d37e1
    }
  ],
  "resources": [
    {
      "name": "cluster.rb",
      "path": "resources/cluster.rb",
      "checksum": "85e74276e19bfdad581dce4f5c59f94a",
      "specificity": "default",
      "url": "https://chef.example/bookshelf/organization-9f69768696feedcd165633b8b475cc0b/checksum-85e74276e19bfdad581
    }
  ],
  "root_files": [
    {
      "name": "metadata.rb",
      "path": "metadata.rb",
      "checksum": "36b395e758138a4295d1e3f9b3df5da9",
      "specificity": "default",
      "url": "https://chef.example/bookshelf/organization-9f69768696feedcd165633b8b475cc0b/checksum-36b395e758138a4295d
    },
    {
      "name": "README.md",
      "path": "README.md",
      "checksum": "99873670f0994642f5e6baade52c8020",
      "specificity": "default",
      "url": "https://chef.example/bookshelf/organization-9f69768696feedcd165633b8b475cc0b/checksum-99873670f0994642f5e
    }
  ],
  "templates": [
    {
      "name": "default.rabbitmq-server.erb",
      "path": "templates/default/default.rabbitmq-server.erb",
      "checksum": "077855f4dc37f7fb708976134d8b2551",
      "specificity": "default",
      "url": "https://chef.example/bookshelf/organization-9f69768696feedcd165633b8b475cc0b/checksum-077855f4dc37f7fb708
    },
    {
      "name": "90forceyes.erb",
      "path": "templates/default/90forceyes.erb",
      "checksum": "73cc571097cf77c74b4e7b5b680020c9",
      "specificity": "default",
      "url": "https://chef.example/bookshelf/organization-9f69768696feedcd165633b8b475cc0b/checksum-73cc571097cf77c74b4
    }
  ],
  "metadata": {
    "name": "rabbitmq",
    "description": "Installs and configures RabbitMQ server",
    "long_description": "",
    "maintainer": "Chef, Inc. and contributors",
    "maintainer_email": "mklishin@pivotal.io",
    "license": "Apache-2.0",
    "platforms": {
      "amazon": ">= 2.0",
      "centos": ">= 7.0",
      "debian": ">= 8.0",
      "opensuse": ">= 0.0.0",
      "opensuseleap": ">= 0.0.0",
      "oracle": ">= 0.0.0",
      "redhat": ">= 0.0.0",
      "scientific": ">= 0.0.0",
      "smartos": ">= 0.0.0",
      "suse": ">= 0.0.0",
      "ubuntu": ">= 14.04"
    },
    "dependencies": {
      "erlang": ">= 0.0.0",
      "yum-epel": ">= 0.0.0",
      "yum-erlang_solutions": ">= 0.0.0",
      "dpkg_autostart": ">= 0.0.0",
      "logrotate": ">= 0.0.0"
    },
    "providing": {
      "rabbitmq::cluster": ">= 0.0.0",
      "rabbitmq::community_plugins": ">= 0.0.0",
      "rabbitmq": ">= 0.0.0",
      "rabbitmq::erlang_package": ">= 0.0.0",
      "rabbitmq::esl_erlang_package": ">= 0.0.0",
      "rabbitmq::management_ui": ">= 0.0.0",
      "rabbitmq::mgmt_console": ">= 0.0.0",
      "rabbitmq::plugin_management": ">= 0.0.0",
      "rabbitmq::plugins": ">= 0.0.0",
      "rabbitmq::policies": ">= 0.0.0",
      "rabbitmq::policy_management": ">= 0.0.0",
      "rabbitmq::systemd_limits": ">= 0.0.0",
      "rabbitmq::user_management": ">= 0.0.0",
      "rabbitmq::users": ">= 0.0.0",
      "rabbitmq::vhosts": ">= 0.0.0",
      "rabbitmq::virtualhost_management": ">= 0.0.0"
    },
    "recipes": {
      "rabbitmq": "Install and configure RabbitMQ",
      "rabbitmq::systemd_limits": "Sets up kernel limits (e.g. nofile) for RabbitMQ via systemd",
      "rabbitmq::cluster": "Set up RabbitMQ clustering.",
      "rabbitmq::management_ui": "Sets up RabbitMQ management plugin/UI",
      "rabbitmq::mgmt_console": "Deprecated, alias for rabbitmq::management_ui",
      "rabbitmq::plugins": "Manage plugins with node attributes",
      "rabbitmq::plugin_management": "Deprecated, alias for rabbitmq::plugins",
      "rabbitmq::vhosts": "Manage virtual hosts with node attributes",
      "rabbitmq::virtualhost_management": "Deprecated, alias for rabbitmq::vhosts",
      "rabbitmq::users": "Manage users with node attributes",
      "rabbitmq::user_management": "Deprecated, alias for rabbitmq::users",
      "rabbitmq::policies": "Manage policies with node attributes",
      "rabbitmq::policy_management": "Deprecated, alias for rabbitmq::policies",
      "rabbitmq::erlang_package": "Provisions Erlang via Team RabbitMQ packages",
      "rabbitmq::esl_erlang_package": "Alias for erlang::esl",
      "rabbitmq::community_plugins": ""
    },
    "version": "5.7.7",
    "source_url": "https://github.com/rabbitmq/chef-cookbook",
    "issues_url": "https://github.com/rabbitmq/chef-cookbook/issues",
    "privacy": false,
    "chef_versions": [
    ],
    "ohai_versions": [
    ],
    "gems": [
    ]
  }
}
```

Response Codes

Response Code	Description
`200`	OK. The request was successful.
`401`	Unauthorized. The user or client who made the request could not be authenticated. Verify the use
`403`	Forbidden. The user who made the request is not authorized to perform the action.
`404`	Not found. The requested object does not exist.

GET

The `GET` method is used to return a single cookbook artifact version.

This method has no parameters.

Request

```
GET /organizations/NAME/cookbook_artifacts/NAME/ID
```

This method has no request body.

Response

The response is similar to:

```
{
  "version": "5.7.7",
  "name": "rabbitmq",
  "identifier": "f3cf8ea7d8bfc59e35ec541946e3e82cd4b73e74",
  "frozen?": false,
  "chef_type": "cookbook_version",
  "attributes": [
    {
      "name": "default.rb",
      "path": "attributes/default.rb",
      "checksum": "e5a530cca3898d8bd07604435dc5156e",
      "specificity": "default",
      "url": "https://chef.example/bookshelf/organization-9f69768696feedcd165633b8b475cc0b/checksum-e5a530cca3898d8bd076
    }
  ],
  "definitions": [
  ],
  "files": [
  ],
  "libraries": [
    {
      "name": "matchers.rb",
      "path": "libraries/matchers.rb",
      "checksum": "24c3f44c4d1d62300a56051f0069f639",
      "specificity": "default",
      "url": "https://chef.example/bookshelf/organization-9f69768696feedcd165633b8b475cc0b/checksum-24c3f44c4d1d62300a56
    },
    {
      "name": "helpers.rb",
      "path": "libraries/helpers.rb",
      "checksum": "df65c4a7259fcb30c6f3f1305ebf7502",
      "specificity": "default",
      "url": "https://chef.example/bookshelf/organization-9f69768696feedcd165633b8b475cc0b/checksum-df65c4a7259fcb30c6f3
    },
    {
      "name": "default.rb",
      "path": "libraries/default.rb",
      "checksum": "94292faac84ba797e720501700b30f74",
      "specificity": "default",
      "url": "https://chef.example/bookshelf/organization-9f69768696feedcd165633b8b475cc0b/checksum-94292faac84ba797e720
    }
  ],
  "providers": [
    {
      "name": "user.rb",
      "path": "providers/user.rb",
      "checksum": "c31c9cc749f21962c825f983a6679d94",
      "specificity": "default",
      "url": "https://chef.example/bookshelf/organization-9f69768696feedcd165633b8b475cc0b/checksum-c31c9cc749f21962c825
    },
    {
      "name": "policy.rb",
      "path": "providers/policy.rb",
      "checksum": "746c8a3f248f5bbfa51f5d2ba60b6315",
      "specificity": "default",
      "url": "https://chef.example/bookshelf/organization-9f69768696feedcd165633b8b475cc0b/checksum-746c8a3f248f5bbfa51f
    }
  ],
  "recipes": [
    {
      "name": "default.rb",
      "path": "recipes/default.rb",
      "checksum": "99a9b404ff6038d6ac55a90ca68c347a",
      "specificity": "default",
      "url": "https://chef.example/bookshelf/organization-9f69768696feedcd165633b8b475cc0b/checksum-99a9b404ff6038d6ac55
    },
    {
      "name": "cluster.rb",
      "path": "recipes/cluster.rb",
      "checksum": "fc0a86c1f858c9d37e11282efc9fe329",
      "specificity": "default",
      "url": "https://chef.example/bookshelf/organization-9f69768696feedcd165633b8b475cc0b/checksum-fc0a86c1f858c9d37e1
    }
  ],
  "resources": [
    {
      "name": "cluster.rb",
      "path": "resources/cluster.rb",
      "checksum": "85e74276e19bfdad581dce4f5c59f94a",
      "specificity": "default",
      "url": "https://chef.example/bookshelf/organization-9f69768696feedcd165633b8b475cc0b/checksum-85e74276e19bfdad581
    }
  ],
  "root_files": [
    {
      "name": "metadata.rb",
      "path": "metadata.rb",
      "checksum": "36b395e758138a4295d1e3f9b3df5da9",
      "specificity": "default",
      "url": "https://chef.example/bookshelf/organization-9f69768696feedcd165633b8b475cc0b/checksum-36b395e758138a4295d
    },
    {
      "name": "README.md",
      "path": "README.md",
      "checksum": "99873670f0994642f5e6baade52c8020",
      "specificity": "default",
      "url": "https://chef.example/bookshelf/organization-9f69768696feedcd165633b8b475cc0b/checksum-99873670f0994642f5e
    }
  ],
  "templates": [
    {
      "name": "default.rabbitmq-server.erb",
      "path": "templates/default/default.rabbitmq-server.erb",
      "checksum": "077855f4dc37f7fb708976134d8b2551",
      "specificity": "default",
      "url": "https://chef.example/bookshelf/organization-9f69768696feedcd165633b8b475cc0b/checksum-077855f4dc37f7fb708
    },
    {
      "name": "90forceyes.erb",
      "path": "templates/default/90forceyes.erb",
      "checksum": "73cc571097cf77c74b4e7b5b680020c9",
      "specificity": "default",
      "url": "https://chef.example/bookshelf/organization-9f69768696feedcd165633b8b475cc0b/checksum-73cc571097cf77c74b4
    }
  ],
  "metadata": {
    "name": "rabbitmq",
    "description": "Installs and configures RabbitMQ server",
    "long_description": "",
    "maintainer": "Chef, Inc. and contributors",
    "maintainer_email": "mklishin@pivotal.io",
    "license": "Apache-2.0",
    "platforms": {
      "amazon": ">= 2.0",
      "centos": ">= 7.0",
      "debian": ">= 8.0",
      "opensuse": ">= 0.0.0",
      "opensuseleap": ">= 0.0.0",
      "oracle": ">= 0.0.0",
      "redhat": ">= 0.0.0",
      "scientific": ">= 0.0.0",
      "smartos": ">= 0.0.0",
      "suse": ">= 0.0.0",
      "ubuntu": ">= 14.04"
    },
    "dependencies": {
      "erlang": ">= 0.0.0",
      "yum-epel": ">= 0.0.0",
      "yum-erlang_solutions": ">= 0.0.0",
      "dpkg_autostart": ">= 0.0.0",
      "logrotate": ">= 0.0.0"
    },
    "providing": {
      "rabbitmq::cluster": ">= 0.0.0",
      "rabbitmq::community_plugins": ">= 0.0.0",
      "rabbitmq": ">= 0.0.0",
      "rabbitmq::erlang_package": ">= 0.0.0",
      "rabbitmq::esl_erlang_package": ">= 0.0.0",
      "rabbitmq::management_ui": ">= 0.0.0",
      "rabbitmq::mgmt_console": ">= 0.0.0",
      "rabbitmq::plugin_management": ">= 0.0.0",
      "rabbitmq::plugins": ">= 0.0.0",
      "rabbitmq::policies": ">= 0.0.0",
      "rabbitmq::policy_management": ">= 0.0.0",
      "rabbitmq::systemd_limits": ">= 0.0.0",
      "rabbitmq::user_management": ">= 0.0.0",
      "rabbitmq::users": ">= 0.0.0",
      "rabbitmq::vhosts": ">= 0.0.0",
      "rabbitmq::virtualhost_management": ">= 0.0.0"
    },
    "recipes": {
      "rabbitmq": "Install and configure RabbitMQ",
      "rabbitmq::systemd_limits": "Sets up kernel limits (e.g. nofile) for RabbitMQ via systemd",
      "rabbitmq::cluster": "Set up RabbitMQ clustering.",
      "rabbitmq::management_ui": "Sets up RabbitMQ management plugin/UI",
      "rabbitmq::mgmt_console": "Deprecated, alias for rabbitmq::management_ui",
      "rabbitmq::plugins": "Manage plugins with node attributes",
      "rabbitmq::plugin_management": "Deprecated, alias for rabbitmq::plugins",
      "rabbitmq::vhosts": "Manage virtual hosts with node attributes",
      "rabbitmq::virtualhost_management": "Deprecated, alias for rabbitmq::vhosts",
      "rabbitmq::users": "Manage users with node attributes",
      "rabbitmq::user_management": "Deprecated, alias for rabbitmq::users",
      "rabbitmq::policies": "Manage policies with node attributes",
      "rabbitmq::policy_management": "Deprecated, alias for rabbitmq::policies",
      "rabbitmq::erlang_package": "Provisions Erlang via Team RabbitMQ packages",
      "rabbitmq::esl_erlang_package": "Alias for erlang::esl",
      "rabbitmq::community_plugins": ""
    },
    "version": "5.7.7",
    "source_url": "https://github.com/rabbitmq/chef-cookbook",
    "issues_url": "https://github.com/rabbitmq/chef-cookbook/issues",
    "privacy": false,
    "chef_versions": [
    ],
    "ohai_versions": [
    ],
    "gems": [
    ]
  }
}
```

Response Codes

Response Code	Description
`200`	OK. The request was successful.
`401`	Unauthorized. The user or client who made the request could not be authenticated. Verify the use
`403`	Forbidden. The user who made the request is not authorized to perform the action.
`404`	Not found. The requested object does not exist.

PUT

The `PUT` method is used to create or update a single cookbook artifact version.

This method has no parameters.

Request

```
PUT /organizations/NAME/cookbook_artifacts/NAME/ID
```

The request body is similar to:

```
{
  "definitions": [
    {
      "name": "unicorn_config.rb",
      "checksum": "c92b659171552e896074caa58dada0c2",
      "path": "definitions/unicorn_config.rb",
      "specificity": "default"
    }
  ],
  "attributes": [],
  "files": [],
  "providers": [],
  "metadata": {
    "dependencies": {"ruby": [], "rubygems": []},
    "name": "unicorn",
    "maintainer_email": "ops@chef.io",
    "attributes": {},
    "license": "Apache 2.0",
    "suggestions": {},
    "platforms": {},
    "maintainer": "Opscode, Inc",
    "long_description": "= LICENSE AND AUTHOR:\\n\\nAuthor:: Adam Jacob...",
    "recommendations": {},
    "version": "0.1.2",
    "conflicting": {},
    "recipes": {"unicorn": "Installs unicorn rubygem"},
    "groupings": {},
    "replacing": {},
    "description": "Installs/Configures unicorn",
    "providing": {}
  },
  "libraries": [],
  "templates": [
    {
      "name": "unicorn.rb.erb",
      "checksum": "36a1cc1b225708db96d48026c3f624b2",
      "path": "templates/default/unicorn.rb.erb",
      "specificity": "default"
    }
  ],
  "resources": [],
  "name": "unicorn",
  "identifier": "ba0dadcbca26710a521e0e3160cc5e20",
  "recipes": [
    {
      "name": "default.rb",
      "checksum": "ba0dadcbca26710a521e0e3160cc5e20",
      "path": "recipes/default.rb",
      "specificity": "default"
    }
  ],
  "root_files": [
    {
      "name": "README.rdoc",
      "checksum": "d18c630c8a68ffa4852d13214d0525a6",
      "path": "README.rdoc",
      "specificity": "default"
    },
    {
      "name": "metadata.rb",
      "checksum": "967087a09f48f234028d3aa27a094882",
      "path": "metadata.rb",
      "specificity": "default"
    },
    {
      "name": "metadata.json",
      "checksum": "45b27c78955f6a738d2d42d88056c57c",
      "path": "metadata.json",
      "specificity": "default"
    }
  ],
  "chef_type": "cookbook_artifact_version"
}
```

where the `checksum` values must have already been uploaded to the Chef Infra Server using the sandbox endpoint. Once a file with a particular checksum has bee be garbage collected.

Response

This method has no response body.

Response Codes

Response Code	Description
`200`	OK. The request was successful.
`401`	Unauthorized. The user or client who made the request could not be authenticated. Verify the use
`403`	Forbidden. The user who made the request is not authorized to perform the action.
`413`	Request entity too large. A request may not be larger than 1000000 bytes.

/cookbooks

{{% cookbooks_summary %}}

When a cookbook is uploaded, only files that are new or updated will be included. This approach minimizes the amount of storage and time that is required during t Infra Client uses a checksum and assigns a checksum to each file. These checksums are used in the cookbook version manifest, alongside the same records that s from which the file's contents can be retrieved.

The `/cookbooks` endpoint has the following methods: `GET`.

GET

The `GET` method is used to return a hash of all cookbooks and cookbook versions.

This method has the following parameters:

Parameter	Description
`num_versions=n`	The number of cookbook versions to include in the response, where `n` is the number of cookbook versions. Fc (newest to oldest). Use `num_versions=all` to return all cookbook versions. If `num_versions` is not specified, a of each cookbook is returned).

Request

```
GET /organizations/NAME/cookbooks
```

Response

The response is similar to:

```
{
  "apache2": {
    "url": "https://localhost/cookbooks/apache2",
    "versions": [
      {"url": "https://localhost/cookbooks/apache2/5.1.0",
       "version": "5.1.0"},
      {"url": "https://localhost/cookbooks/apache2/4.2.0",
       "version": "4.2.0"}
    ]
  },
  "nginx": {
    "url": "https://localhost/cookbooks/nginx",
    "versions": [
      {"url": "https://localhost/cookbooks/nginx/1.0.0",
       "version": "1.0.0"},
      {"url": "https://localhost/cookbooks/nginx/0.3.0",
       "version": "0.3.0"}
    ]
  }
}
```

Response Codes

Response Code	Description
`200`	OK. The request was successful.
`401`	Unauthorized. The user or client who made the request could not be authenticated. Verify the use

Response Code	Description
`403`	Forbidden. The user who made the request is not authorized to perform the action.

/cookbooks/_latest

The `/cookbooks/_latest` endpoint has the following methods: `GET`.

GET

The `GET` method is used to return a list of the most recent cookbook versions.

This method has no parameters.

Request

```
GET /organizations/NAME/cookbooks/_latest
```

Response

For example, if cookbooks `foo` and `bar` both exist on the Chef Infra Server and both with versions `0.1.0` and `0.2.0`, the response is similar to:

```
{
  "foo": "https://localhost/cookbooks/foo/0.2.0",
  "bar": "https://localhost/cookbooks/bar/0.2.0"
}
```

Response Codes

Response Code	Description
`200`	OK. The request was successful.
`401`	Unauthorized. The user or client who made the request could not be authenticated. Verify the use
`403`	Forbidden. The user who made the request is not authorized to perform the action.
`404`	Not found. The requested object does not exist.

/cookbooks/_recipes

The `/cookbooks/_recipes` endpoint has the following methods: `GET`.

GET

The `GET` method is used to return the names of all recipes in the most recent cookbook versions.

This method has no parameters.

Request

```
GET /organizations/NAME/cookbooks/_recipes
```

Response

The response is similar to:

```
{
}
```

Response Codes

Response Code	Description
`200`	OK. The request was successful.
`401`	Unauthorized. The user or client who made the request could not be authenticated. Verify the use
`403`	Forbidden. The user who made the request is not authorized to perform the action.
`404`	Not found. The requested object does not exist.

/cookbooks/NAME

The `/cookbooks/NAME` endpoint has the following methods: `GET`.

GET

The `GET` method is used to return a hash that contains a key-value pair that corresponds to the specified cookbook, with a URL for the cookbook and for each versi

Request

```
GET /organizations/NAME/cookbooks/NAME
```

Response

The response is similar to:

```
{
  "apache2": {
    "url": "https://localhost/cookbooks/apache2",
    "versions": [
      {"url": "https://localhost/cookbooks/apache2/5.1.0",
       "version": "5.1.0"},
      {"url": "https://localhost/cookbooks/apache2/4.2.0",
       "version": "4.2.0"}
    ]
  }
}
```

Response Codes

Response Code	Description
`200`	OK. The request was successful.
`401`	Unauthorized. The user or client who made the request could not be authenticated. Verify the use
`403`	Forbidden. The user who made the request is not authorized to perform the action.
`404`	Not found. The requested object does not exist.

/cookbooks/NAME/version

{{% cookbooks_version %}}

The `/cookbooks/NAME/VERSION` endpoint has the following methods: `DELETE`, `GET`, and `PUT`.

DELETE

The `DELETE` method is used to delete a cookbook version.

This method has no parameters.

Request

```
DELETE /organizations/NAME/cookbooks/NAME/VERSION
```

Response

This method has no response body. Unused `checksum` values will be garbage collected.

Response Codes

Response Code	Description
`200`	OK. The request was successful.
`401`	Unauthorized. The user or client who made the request could not be authenticated. Verify the use
`403`	Forbidden. The user who made the request is not authorized to perform the action.
`404`	Not found. The requested object does not exist.

GET

The `GET` method is used to return a description of a cookbook, including its metadata and links to component files.

This method has no parameters.

Request

```
GET /organizations/NAME/cookbooks/NAME/VERSION
```

where `VERSION` can be `_latest` in order to float to head.

Response

The response is similar to:

```
{
  "cookbook_name": "getting-started",
  "files": [

],
"chef_type": "cookbook_version",
"definitions": [


],
"libraries": [


],
"attributes": [
{
"url": "https://domain.com/org_name/(...rest of URL)",
"path": "attributes/default.rb",
"specificity": "default",
"name": "default.rb",
"checksum": "fa0fc4abf3f6787fdsaasadfrc5c35de667c"
}
],
"recipes": [
{
"url": "https://domain.com/org_name/(...rest of URL)",
"path": "recipes/default.rb",
"specificity": "default",
"name": "default.rb",
"checksum": "7e79b1ace7728fdsadfsdaf857e60fc69"
}
],
"providers": [


],
"resources": [


],
"templates": [
{
"url": "https://domain.com/org_name/(...rest of URL)",
"path": "templates/default/chef-getting-started.txt.erb",
"specificity": "default",
"name": "chef-getting-started.txt.erb",
"checksum": "a29d6f2545sdffds1f140c3a78b1fe"
}
],
"root_files": [
{
"url": "https://domain.com/org_name/(...rest of URL)",
"path": ".DS_Store",
"specificity": "default",
"name": ".DS_Store",
"checksum": "c107b500aafd12asdffdsdf5c2a7d6"
},
{
"url": "https://domain.com/org_name/(...rest of URL)",
"path": "metadata.json",
"specificity": "default",
"name": "metadata.json",
"checksum": "20f09570e54dasdf0f3ae01e6401c90f"
},
{
"url": "https://domain.com/org_name/(...rest of URL)",
"path": "metadata.rb",
"specificity": "default",
"name": "metadata.rb",
"checksum": "71027aefasd487fdsa4cb6994b66ed"
},
{
"url": "https://domain.com/org_name/(...rest of URL)",
"path": "README.rdoc",
"specificity": "default",
"name": "README.rdoc",
"checksum": "8b9275e56fee974easdfasdfbb729"
}
],
"name": "getting-started-0.4.0",
"frozen?": false,
"version": "0.4.0",
"json_class": "Chef::CookbookVersion",
"metadata": {
"maintainer": "Maintainer",
"attributes": { },
"suggestions": { },
"recipes": { "getting-started": "" },
"dependencies": { },
"platforms": { },
"groupings": { },
"recommendations": { },
"name": "getting-started",
"description": "description",
"version": "0.4.0",
"maintainer_email": "sysadmin@opscode.com",
"long_description": "= DESCRIPTION:\n\nThis cookbook is used to do some things.\n\n",
"providing": { "getting-started": ">= 0.0.0" },
"replacing": { },
"conflicting": { },
"license": "Apache 2.0"
}
}
```

Response Codes

Response Code	Description
`200`	OK. The request was successful.
`401`	Unauthorized. The user or client who made the request could not be authenticated. Verify the use
`403`	Forbidden. The user who made the request is not authorized to perform the action.
`404`	Not found. The requested object does not exist.

PUT

The `PUT` method is used to create or update a cookbook version.

This method has no parameters.

Request

```
PUT /organizations/NAME/cookbooks/NAME/VERSION
```

with a request body similar to:

```
{
  "definitions": [
    {
      "name": "unicorn_config.rb",
      "checksum": "c92b659171552e896074caa58dada0c2",
      "path": "definitions/unicorn_config.rb",
      "specificity": "default"
    }
  ],
  "name": "unicorn-0.1.2",
  "attributes": [],
  "files": [],
  "json_class": "Chef::CookbookVersion",
  "providers": [],
  "metadata": {
    "dependencies": {"ruby": [], "rubygems": []},
    "name": "unicorn",
    "maintainer_email": "ops@opscode.com",
    "attributes": {},
    "license": "Apache 2.0",
    "suggestions": {},
    "platforms": {},
    "maintainer": "Opscode, Inc",
    "long_description": "= LICENSE AND AUTHOR:\n\nAuthor:: Adam Jacob...",
    "recommendations": {},
    "version": "0.1.2",
    "conflicting": {},
    "recipes": {"unicorn": "Installs unicorn rubygem"},
    "groupings": {},
    "replacing": {},
    "description": "Installs/Configures unicorn",
    "providing": {}
  },
  "libraries": [],
  "templates": [
    {
      "name": "unicorn.rb.erb",
      "checksum": "36a1cc1b225708db96d48026c3f624b2",
      "path": "templates/default/unicorn.rb.erb",
      "specificity": "default"
    }
  ],
  "resources": [],
  "cookbook_name": "unicorn",
  "version": "0.1.2",
  "recipes": [
    {
      "name": "default.rb",
      "checksum": "ba0dadcbca26710a521e0e3160cc5e20",
      "path": "recipes/default.rb",
      "specificity": "default"
    }
  ],
  "root_files": [
    {
      "name": "README.rdoc",
      "checksum": "d18c630c8a68ffa4852d13214d0525a6",
      "path": "README.rdoc",
      "specificity": "default"
    },
    {
      "name": "metadata.rb",
      "checksum": "967087a09f48f234028d3aa27a094882",
      "path": "metadata.rb",
      "specificity": "default"
    },
    {
      "name": "metadata.json",
      "checksum": "45b27c78955f6a738d2d42d88056c57c",
      "path": "metadata.json",
      "specificity": "default"
    }
  ],
  "chef_type": "cookbook_version"
}
```

where the `checksum` values must have already been uploaded to the Chef Infra Server using the sandbox endpoint. Once a file with a particular checksum has bee be garbage collected.

Response

This method has no response body.

Response Codes

Response Code	Description
`200`	OK. The request was successful.
`401`	Unauthorized. The user or client who made the request could not be authenticated. Verify the use
`403`	Forbidden. The user who made the request is not authorized to perform the action.
`413`	Request entity too large. A request may not be larger than 1000000 bytes.

/data

{{% data_bag %}}

The `/data` endpoint has the following methods: `GET` and `POST`.

GET

The `GET` method is used to return a list of all data bags on the Chef Infra Server.

This method has no parameters.

Request

```
GET /organizations/NAME/data
```

Response

The response is similar to:

```
{
  "users": "https://chef.example/organizations/NAME/data/users",
  "applications": "https://chef.example/organizations/NAME/data/applications"
}
```

shown as a list of key-value pairs, where (in the example above) `users` and `applications` are the names of data bags and `https://chef.example/organizations/I`

Response Codes

Response Code	Description
`200`	OK. The request was successful.
`401`	Unauthorized. The user or client who made the request could not be authenticated. Verify the use
`403`	Forbidden. The user who made the request is not authorized to perform the action.

POST

The `POST` method is used to create a new data bag on the Chef Infra Server.

This method has no parameters.

Request

```
POST /organizations/NAME/data
```

with a request body that contains the key-value pair for the data bag and is similar to:

```
{
  "name": "users"
}
```

where (in the example above) `name` is the key and "users" is the value.

Response

The response is similar to:

```
{
   "uri": "https://organizations/NAME/data/users",
}
```

Response Codes

Response Code	Description
`201`	Created. The object was created.
`400`	Bad request. The contents of the request are not formatted correctly.
`401`	Unauthorized. The user or client who made the request could not be authenticated. Verify the use
`403`	Forbidden. The user who made the request is not authorized to perform the action.
`409`	Conflict. A databag with that name already exists.
`413`	Request entity too large. A request may not be larger than 1000000 bytes.

/data/NAME

The `/data/NAME` endpoint is used to view and update data for a specific data bag. This endpoint has the following methods: `DELETE`, `GET`, and `POST`.

DELETE

The `DELETE` method is used to delete a data bag.

This method has no parameters.

Request

```
DELETE /organizations/NAME/data/NAME
```

This method has no request body.

Response

The response is similar to:

```
{
  "name": "users",
  "json_class": "Chef::DataBag",
  "chef_type": "data_bag"
}
```

where the key-value pairs represent the last state of the data bag item.

Response Codes

Response Code	Description
`200`	OK. The request was successful.
`401`	Unauthorized. The user or client who made the request could not be authenticated. Verify the use
`403`	Forbidden. The user who made the request is not authorized to perform the action.
`404`	Not found. The requested object does not exist.

GET

The `GET` method is used to return a hash of all entries in the specified data bag.

This method has no parameters.

Request

```
GET /organizations/NAME/data/NAME
```

Response

The response is similar to:

```
{
   "adam": "https://chef.example/organizations/NAME/data/users/adam"
}
```

Response Codes

Response Code	Description
`200`	OK. The request was successful.
`401`	Unauthorized. The user or client who made the request could not be authenticated. Verify the use
`403`	Forbidden. The user who made the request is not authorized to perform the action.
`404`	Not found. The requested object does not exist.

POST

The `POST` method is used to create a new data bag item.

This method has no parameters.

Request

```
POST /organizations/NAME/data/NAME
```

with a request body similar to:

```
{
  "id": "adam",
  "real_name": "Adam Jacob"
}
```

where `id` is required.

Response

This method has no response body.

Response Codes

Response Code	Description
`201`	OK. The item was created.
`400`	Bad request. The contents of the request are not formatted correctly.
`401`	Unauthorized. The user or client who made the request could not be authenticated. Verify the use
`403`	Forbidden. The user who made the request is not authorized to perform the action.
`404`	Not found. The requested object does not exist.
`409`	Conflict. The object already exists.
`413`	Request entity too large. A request may not be larger than 1000000 bytes.

/data/NAME/ITEM

{{% data_bag_item %}}

The `/data/NAME/ITEM` endpoint allows the key-value pairs within a data bag item to be viewed and managed. This endpoint has the following methods: `DELETE`, `GET`

DELETE

The `DELETE` method is used to delete a key-value pair in a data bag item.

This method has no parameters.

Request

```
DELETE /organizations/NAME/data/NAME/ITEM
```

Response

The response is similar to:

```
{
}
```

Response Codes

Response Code	Description
`200`	OK. The request was successful.
`401`	Unauthorized. The user or client who made the request could not be authenticated. Verify the use
`403`	Forbidden. The user who made the request is not authorized to perform the action.
`404`	Not found. The requested object does not exist.

GET

The `GET` method is used to view all of the key-value pairs in a data bag item.

This method has no parameters.

Request

```
GET /organizations/NAME/data/NAME/ITEM
```

Response

The response is similar to:

```
{
  "real_name": "Adam Jacob",
  "id": "adam"
}
```

Response Codes

Response Code	Description
`200`	OK. The request was successful.
`401`	Unauthorized. The user or client who made the request could not be authenticated. Verify the use
`403`	Forbidden. The user who made the request is not authorized to perform the action.
`404`	Not found. The requested object does not exist.

PUT

The `PUT` method is used to replace the contents of a data bag item with the contents of this request.

This method has no parameters.

Request

```
PUT /organizations/NAME/data/NAME/ITEM
```

with a request body similar to:

```
{
  "real_name": "Adam Brent Jacob",
  "id": "adam"
}
```

where `id` is required.

Response

The response is similar to:

```
{
}
```

Response Codes

Response Code	Description
`200`	OK. The request was successful.
`401`	Unauthorized. The user or client who made the request could not be authenticated. Verify the use
`403`	Forbidden. The user who made the request is not authorized to perform the action.
`404`	Not found. The requested object does not exist.
`413`	Request entity too large. A request may not be larger than 1000000 bytes.

/environments

{{% environment %}}

The `/environments` endpoint has the following methods: `GET` and `POST`.

GET

The `GET` method is used to return a data structure that contains a link to each available environment.

This method has no parameters.

Request

```
GET /organizations/NAME/environments
```

Response

The response is similar to:

```
{
  "_default": "https://api.opscode.com/organizations/org_name/environments/_default",
  "webserver": "https://api.opscode.com/organizations/org_name/environments/webserver"
}
```

Response Codes

Response Code	Description
`200`	OK. The request was successful.
`401`	Unauthorized. The user or client who made the request could not be authenticated. Verify the use
`403`	Forbidden. The user who made the request is not authorized to perform the action.

POST

The `POST` method is used to create a new environment.

This method has no parameters.

Request

```
POST /organizations/NAME/environments
```

with a request body similar to:

```
{
  "name": "dev",
  "default_attributes": {},
  "json_class": "Chef::Environment",
  "description": "",
  "cookbook_versions": {},
  "chef_type": "environment"
}
```

Response

The response is similar to:

```
{ "uri": "https://localhost/environments/dev" }
```

Response Codes

Response Code	Description
`201`	Created. The object was created.
`400`	Bad request. The contents of the request are not formatted correctly.
`401`	Unauthorized. The user or client who made the request could not be authenticated. Verify the use

Response Code	Description
`403`	Forbidden. The user who made the request is not authorized to perform the action.
`409`	Conflict. The object already exists.
`413`	Request entity too large. A request may not be larger than 1000000 bytes.

/environments/_default

The `/environments/_default` endpoint has the following methods: `GET`.

GET

The `GET` method is used to get information about the `_default` environment on the Chef Infra Server.

This method has no parameters.

Request

```
GET /organizations/NAME/environments/_default
```

Response

The response is similar to:

```
{
  "name": "_default",
  "description": "The default Chef environment",
  "json_class": "Chef::Environment",
  "chef_type": "environment",
  "default_attributes": {

},
"override_attributes": {


},
"cookbook_versions": {


}
}
```

Response Codes

Response Code	Description
`200`	OK. The request was successful.
`401`	Unauthorized. The user or client who made the request could not be authenticated. Verify the use
`403`	Forbidden. The user who made the request is not authorized to perform the action.
`404`	Not found. The requested object does not exist.

/environments/NAME

The `/environments/NAME` endpoint has the following methods: `DELETE`, `GET`, and `PUT`.

DELETE

The `DELETE` method is used to delete an environment.

This method has no parameters.

Request

```
DELETE /organizations/NAME/environments/NAME
```

Response

The response will return the JSON for the environment that was deleted, similar to:

```
{
  "name":"backend",
  "description":"",
  "cookbook_versions":{},
  "json_class":"Chef::Environment",
  "chef_type":"environment",
  "default_attributes":{},
  "override_attributes":{}
}
```

Response Codes

Response Code	Description
`200`	OK. The request was successful.
`401`	Unauthorized. The user or client who made the request could not be authenticated. Verify the use
`403`	Forbidden. The user who made the request is not authorized to perform the action.
`404`	Not found. The requested object does not exist.

GET

The `GET` method is used to return the details for an environment as JSON.

This method has no parameters.

Request

```
GET /organizations/NAME/environments/NAME
```

Response

The response is similar to:

```
{
  "name": "_default",
  "description": "The default Chef environment",
  "json_class": "Chef::Environment",
  "chef_type": "environment",
  "default_attributes": { }
  "override_attributes": { },
  "cookbook_versions": { },
}
```

Response Codes

Response Code	Description
`200`	OK. The request was successful.
`401`	Unauthorized. The user or client who made the request could not be authenticated. Verify the use
`403`	Forbidden. The user who made the request is not authorized to perform the action.
`404`	Not found. The requested object does not exist.

PUT

The `PUT` method is used to update the details of an environment on the Chef Infra Server.

This method has no parameters.

Request

```
PUT /organizations/NAME/environments/NAME
```

with a request body that contains the updated JSON for the environment and is similar to:

```
{
  "name": "dev",
  "attributes": {},
  "json_class": "Chef::Environment",
  "description": "The Dev Environment",
  "cookbook_versions": {},
  "chef_type": "environment"
}
```

Response

The response will return the updated environment.

Response Codes

Response Code	Description
`200`	OK. The request was successful.
`401`	Unauthorized. The user or client who made the request could not be authenticated. Verify the use
`403`	Forbidden. The user who made the request is not authorized to perform the action.
`404`	Not found. The requested object does not exist.
`413`	Request entity too large. A request may not be larger than 1000000 bytes.

/environments/NAME/cookbooks/NAME

The `/environments/NAME/cookbooks/NAME` endpoint has the following methods: `GET`.

GET

The `GET` method is used to return a hash of key-value pairs for the requested cookbook.

This method has the following parameters:

Parameter	Description
`num_versions=n`	The number of cookbook versions to include in the response, where `n` is the number of cookbook versions. Fc (newest to oldest). Use `num_versions=all` to return all cookbook versions. If `num_versions` is not specified, a of each cookbook is returned).

Request

```
GET /organizations/NAME/environments/NAME/cookbooks/NAME
```

where the first instance of `NAME` is the name of the environment, and the second instance is the name of the cookbook.

Response

The response is similar to:

```
{
  "apache2": {
    "url": "https://localhost/cookbooks/apache2",
    "versions": [
      {"url": "https://localhost/cookbooks/apache2/5.1.0",
       "version": "5.1.0"},
      {"url": "https://localhost/cookbooks/apache2/4.2.0",
       "version": "4.2.0"}
    ]
  }
}
```

Response Codes

Response Code	Description
`200`	OK. The request was successful.
`401`	Unauthorized. The user or client who made the request could not be authenticated. Verify the use
`403`	Forbidden. The user who made the request is not authorized to perform the action.
`404`	Not found. The requested object does not exist.

/environments/NAME/cookbook_versions

The `/environments/NAME/cookbook_versions` endpoint has the following methods: `POST`.

POST

The `POST` method is used to return a hash of the cookbooks and cookbook versions (including all dependencies) that are required by the `run_list` array. Versior
Version constraints may also be present when the `cookbook_versions` attributes is specified for an environment or when dependencies are specified by a cookbook

This method has no parameters.

Request

```
POST /organizations/NAME/environments/NAME/cookbook_versions
```

with a request body similar to:

```
{
  "run_list": [
    "zed@0.0.1",
    "bar",
    "mysql",
    "gem",
    "nginx@0.99.2",
    "cron",
    "foo"
  ]
}
```

where `@x.x.x` represents a constraint for a cookbook version.

Response

The response will return a list of cookbooks that are required by the `run_list` array contained in the request. The cookbooks that are returned are often the late
dependencies a cookbook may have for specific cookbook versions, a request may not always return the latest cookbook version for each cookbook.

The response is similar to:

```
{
  "cookbook_name": {
    "recipes": [
      {
        "name": "default.rb",
        "path": "recipes/default.rb",
        "checksum": "12345efg78912346abcddefg789",
        "specificity": "default",
        "url": "https://URL"
      },
      {
        "name": "recipe_name.rb",
        "path": "recipes/recipe_name.rb",
        "checksum": "12345efg78912346abcddefg789",
        "specificity": "default",
        "url": "https://URL"
      },
      {
        ...
      }
    ],
    "definitions": [
```

```
],
<span class="hljs-string">"libraries"</span>: [

],
<span class="hljs-string">"attributes"</span>: [

],
<span class="hljs-string">"files"</span>: [

],
<span class="hljs-string">"templates"</span>: [
  {
    <span class="hljs-string">"name"</span>: <span class="hljs-string">"template_name.erb"</span>,
    <span class="hljs-string">"path"</span>: <span class="hljs-string">"templates/default/template_name.erb"</span>,
    <span class="hljs-string">"checksum"</span>: <span class="hljs-string">"12345efg78912346abcddefg789"</span>,
    <span class="hljs-string">"specificity"</span>: <span class="hljs-string">"default"</span>,
    <span class="hljs-string">"url"</span>: <span class="hljs-string">"https://URL"</span>
  },
  {
    ...
  }
],
<span class="hljs-string">"resources"</span>: [

],
<span class="hljs-string">"providers"</span>: [

],
<span class="hljs-string">"root_files"</span>: [
  {
    <span class="hljs-string">"name"</span>: <span class="hljs-string">"metadata.rb"</span>,
    <span class="hljs-string">"path"</span>: <span class="hljs-string">"metadata.rb"</span>,
    <span class="hljs-string">"checksum"</span>: <span class="hljs-string">"12345efg78912346abcddefg789"</span>,
    <span class="hljs-string">"specificity"</span>: <span class="hljs-string">"default"</span>,
    <span class="hljs-string">"url"</span>: <span class="hljs-string">"https://URL"</span>
  }
],
<span class="hljs-string">"cookbook_name"</span>: <span class="hljs-string">"cookbook_name-1.0.2"</span>,
<span class="hljs-string">"metadata"</span>: {
  <span class="hljs-string">"name"</span>: <span class="hljs-string">"cookbook_name"</span>,
  <span class="hljs-string">"description"</span>: <span class="hljs-string">"description"</span>,
  <span class="hljs-string">"long_description"</span>: <span class="hljs-string">""</span>,
  <span class="hljs-string">"maintainer"</span>: <span class="hljs-string">"maintainer"</span>,
  <span class="hljs-string">"maintainer_email"</span>: <span class="hljs-string">"maintainer@email.com"</span>,
  <span class="hljs-string">"license"</span>: <span class="hljs-string">"license"</span>,
  <span class="hljs-string">"platforms"</span>: {
  },
  <span class="hljs-string">"dependencies"</span>: {
    <span class="hljs-string">"cookbook_name"</span>: <span class="hljs-string">"&gt;= 0.0.0"</span>,
    <span class="hljs-string">"cookbook_name"</span>: <span class="hljs-string">"&gt;= 1.2.3"</span>,
    ...
    <span class="hljs-string">"cookbook_name"</span>: <span class="hljs-string">"&gt;= 0.1.0"</span>
  },
  <span class="hljs-string">"recommendations"</span>: {
  },
  <span class="hljs-string">"suggestions"</span>: {
  },
  <span class="hljs-string">"conflicting"</span>: {
  },
  <span class="hljs-string">"providing"</span>: {
    <span class="hljs-string">"cookbook_name"</span>: <span class="hljs-string">"&gt;= 0.0.0"</span>,
    <span class="hljs-string">"cookbook_name::recipe_name"</span>: <span class="hljs-string">"&gt;= 0.0.0"</span>,
    <span class="hljs-string">"cookbook_name::recipe_name"</span>: <span class="hljs-string">"&gt;= 1.2.3"</span>,
    <span class="hljs-string">"cookbook_name::recipe_name"</span>: <span class="hljs-string">"&gt;= 0.1.0"</span>
  },
  <span class="hljs-string">"replacing"</span>: {
  },
  <span class="hljs-string">"attributes"</span>: {
  },
  <span class="hljs-string">"groupings"</span>: {
  },
  <span class="hljs-string">"recipes"</span>: {
    <span class="hljs-string">"cookbook_name"</span>: <span class="hljs-string">"description"</span>,
    <span class="hljs-string">"cookbook_name::recipe_name"</span>: <span class="hljs-string">""</span>,
    ...
    <span class="hljs-string">"cookbook_name::recipe_name"</span>: <span class="hljs-string">""</span>
  },
  <span class="hljs-string">"version"</span>: <span class="hljs-string">"0.0.0"</span>
},
<span class="hljs-string">"version"</span>: <span class="hljs-string">"0.0.0"</span>,
<span class="hljs-string">"name"</span>: <span class="hljs-string">"cookbook_name-1.0.2"</span>,
<span class="hljs-string">"frozen?"</span>: <span class="hljs-literal">false</span>,
<span class="hljs-string">"chef_type"</span>: <span class="hljs-string">"cookbook_version"</span>,
<span class="hljs-string">"json_class"</span>: <span class="hljs-string">"Chef::CookbookVersion"</span>
```

```
},
"cookbook_name": {
"recipes": [
{
"name": "default.rb",
"path": "recipes/default.rb",
"checksum": "12345efg78912346abcddefg789",
"specificity": "default",
"url": "https://URL"
},
],
"definitions": [
```

```
  ],
  <span class="hljs-string">"libraries"</span>: [
    {
      <span class="hljs-string">"name"</span>: <span class="hljs-string">"library_file.rb"</span>,
      <span class="hljs-string">"path"</span>: <span class="hljs-string">"libraries/library_file.rb"</span>,
      <span class="hljs-string">"checksum"</span>: <span class="hljs-string">"12345efg78912346abcddefg789"</span>,
      <span class="hljs-string">"specificity"</span>: <span class="hljs-string">"default"</span>,
      <span class="hljs-string">"url"</span>: <span class="hljs-string">"https://URL"</span>
    }
  ],
  <span class="hljs-string">"attributes"</span>: [
    {
      <span class="hljs-string">"name"</span>: <span class="hljs-string">"default.rb"</span>,
      <span class="hljs-string">"path"</span>: <span class="hljs-string">"attributes/default.rb"</span>,
      <span class="hljs-string">"checksum"</span>: <span class="hljs-string">"12345efg78912346abcddefg789"</span>,
      <span class="hljs-string">"specificity"</span>: <span class="hljs-string">"default"</span>,
      <span class="hljs-string">"url"</span>: <span class="hljs-string">"https://URL"</span>
    }
  ],
  <span class="hljs-string">"files"</span>: [

  ],
  <span class="hljs-string">"templates"</span>: [

  ],
  <span class="hljs-string">"resources"</span>: [

  ],
  <span class="hljs-string">"providers"</span>: [

  ],
  <span class="hljs-string">"root_files"</span>: [
    {
      <span class="hljs-string">"name"</span>: <span class="hljs-string">".gitignore"</span>,
      <span class="hljs-string">"path"</span>: <span class="hljs-string">".gitignore"</span>,
      <span class="hljs-string">"checksum"</span>: <span class="hljs-string">"12345efg78912346abcddefg789"</span>,
      <span class="hljs-string">"specificity"</span>: <span class="hljs-string">"default"</span>,
      <span class="hljs-string">"url"</span>: <span class="hljs-string">"https://URL"</span>
    },
    {
      <span class="hljs-string">"name"</span>: <span class="hljs-string">".kitchen.yml"</span>,
      <span class="hljs-string">"path"</span>: <span class="hljs-string">".kitchen.yml"</span>,
      <span class="hljs-string">"checksum"</span>: <span class="hljs-string">"12345efg78912346abcddefg789"</span>,
      <span class="hljs-string">"specificity"</span>: <span class="hljs-string">"default"</span>,
      <span class="hljs-string">"url"</span>: <span class="hljs-string">"https://URL"</span>
    },
    {
      <span class="hljs-string">"name"</span>: <span class="hljs-string">"CHANGELOG.md"</span>,
      <span class="hljs-string">"path"</span>: <span class="hljs-string">"CHANGELOG.md"</span>,
      <span class="hljs-string">"checksum"</span>: <span class="hljs-string">"12345efg78912346abcddefg789"</span>,
      <span class="hljs-string">"specificity"</span>: <span class="hljs-string">"default"</span>,
      <span class="hljs-string">"url"</span>: <span class="hljs-string">"https://URL"</span>
    },
    {
      <span class="hljs-string">"name"</span>: <span class="hljs-string">"CONTRIBUTING"</span>,
      <span class="hljs-string">"path"</span>: <span class="hljs-string">"CONTRIBUTING"</span>,
      <span class="hljs-string">"checksum"</span>: <span class="hljs-string">"12345efg78912346abcddefg789"</span>,
      <span class="hljs-string">"specificity"</span>: <span class="hljs-string">"default"</span>,
      <span class="hljs-string">"url"</span>: <span class="hljs-string">"https://URL"</span>
    },
    {
      <span class="hljs-string">"name"</span>: <span class="hljs-string">"LICENSE"</span>,
      <span class="hljs-string">"path"</span>: <span class="hljs-string">"LICENSE"</span>,
      <span class="hljs-string">"checksum"</span>: <span class="hljs-string">"12345efg78912346abcddefg789"</span>,
      <span class="hljs-string">"specificity"</span>: <span class="hljs-string">"default"</span>,
      <span class="hljs-string">"url"</span>: <span class="hljs-string">"https://URL"</span>
    },
    {
      <span class="hljs-string">"name"</span>: <span class="hljs-string">"metadata.json"</span>,
      <span class="hljs-string">"path"</span>: <span class="hljs-string">"metadata.json"</span>,
      <span class="hljs-string">"checksum"</span>: <span class="hljs-string">"12345efg78912346abcddefg789"</span>,
      <span class="hljs-string">"specificity"</span>: <span class="hljs-string">"default"</span>,
      <span class="hljs-string">"url"</span>: <span class="hljs-string">"https://URL"</span>
    },
    {
      <span class="hljs-string">"name"</span>: <span class="hljs-string">"metadata.rb"</span>,
      <span class="hljs-string">"path"</span>: <span class="hljs-string">"metadata.rb"</span>,
      <span class="hljs-string">"checksum"</span>: <span class="hljs-string">"12345efg78912346abcddefg789"</span>,
      <span class="hljs-string">"specificity"</span>: <span class="hljs-string">"default"</span>,
      <span class="hljs-string">"url"</span>: <span class="hljs-string">"https://URL"</span>
    },
    {
      <span class="hljs-string">"name"</span>: <span class="hljs-string">"README.md"</span>,
      <span class="hljs-string">"path"</span>: <span class="hljs-string">"README.md"</span>,
      <span class="hljs-string">"checksum"</span>: <span class="hljs-string">"12345efg78912346abcddefg789"</span>,
      <span class="hljs-string">"specificity"</span>: <span class="hljs-string">"default"</span>,
      <span class="hljs-string">"url"</span>: <span class="hljs-string">"https://URL"</span>
    },
  ],
  <span class="hljs-string">"chef_type"</span>: <span class="hljs-string">"cookbook_version"</span>,
  <span class="hljs-string">"name"</span>: <span class="hljs-string">"cookbook_name-1.0.2"</span>,
  <span class="hljs-string">"cookbook_name"</span>: <span class="hljs-string">"cookbook_name"</span>,
  <span class="hljs-string">"version"</span>: <span class="hljs-string">"1.0.2"</span>,
  <span class="hljs-string">"metadata"</span>: {
    <span class="hljs-string">"name"</span>: <span class="hljs-string">"cookbook_name"</span>,
    <span class="hljs-string">"description"</span>: <span class="hljs-string">"description"</span>,
    <span class="hljs-string">"long_description"</span>: <span class="hljs-string">""</span>,
    <span class="hljs-string">"maintainer"</span>: <span class="hljs-string">"maintainer"</span>,
    <span class="hljs-string">"maintainer_email"</span>: <span class="hljs-string">"maintainer@email.com"</span>,
    <span class="hljs-string">"license"</span>: <span class="hljs-string">"license"</span>,
    <span class="hljs-string">"platforms"</span>: {
    },
    <span class="hljs-string">"dependencies"</span>: {
    },
    <span class="hljs-string">"recommendations"</span>: {
    },
    <span class="hljs-string">"suggestions"</span>: {
    },
    <span class="hljs-string">"conflicting"</span>: {
    },
    <span class="hljs-string">"providing"</span>: {
    },
    <span class="hljs-string">"replacing"</span>: {
    },
    <span class="hljs-string">"attributes"</span>: {
    },
    <span class="hljs-string">"groupings"</span>: {
    },
    <span class="hljs-string">"recipes"</span>: {
    },
    <span class="hljs-string">"version"</span>: <span class="hljs-string">"1.0.2"</span>
  },
  <span class="hljs-string">"frozen?"</span>: <span class="hljs-literal">true</span>,
  <span class="hljs-string">"json_class"</span>: <span class="hljs-string">"Chef::CookbookVersion"</span>
```

```
},
"cookbook_name": {
...
}
}
```

Response Codes

Response Code	Description
`200`	OK. The request was successful.
`400`	Bad request. The contents of the request are not formatted correctly.
`401`	Unauthorized. The user or client who made the request could not be authenticated. Verify the user/client name
`403`	Forbidden. The user who made the request is not authorized to perform the action.
`404`	Not found. The requested object does not exist.
`412`	Not allowed. A set of cookbooks and/or cookbook versions could not be found that met all of the requirements for a cookbook that does not exist. A constraint on a cookbook made by a run-list, environment, or cookbook
`413`	Request entity too large. A request may not be larger than 1000000 bytes.

/environments/NAME/cookbooks

The `/environments/NAME/cookbooks` endpoint has the following methods: `GET`.

GET

The `GET` method is used to get a list of cookbooks and cookbook versions that are available to the specified environment.

This method has the following parameters:

Parameter	Description
`num_versions=n`	The number of cookbook versions to include in the response, where `n` is the number of cookbook versions. Fc (newest to oldest). Use `num_versions=all` to return all cookbook versions. If `num_versions` is not specified, a of each cookbook is returned).

Request

```
GET /organizations/NAME/environments/NAME/cookbooks
```

Response

The response is similar to:

```
{
  "apache2": {
    "url": "https://localhost/cookbooks/apache2",
    "versions": [
      {"url": "https://localhost/cookbooks/apache2/5.1.0",
       "version": "5.1.0"},
      {"url": "https://localhost/cookbooks/apache2/4.2.0",
       "version": "4.2.0"}
    ]
  },
  "nginx": {
    "url": "https://localhost/cookbooks/nginx",
    "versions": [
      {"url": "https://localhost/cookbooks/nginx/1.0.0",
       "version": "1.0.0"},
      {"url": "https://localhost/cookbooks/nginx/0.3.0",
       "version": "0.3.0"}
    ]
  }
}
```

Response Codes

Response Code	Description
`200`	OK. The request was successful.
`401`	Unauthorized. The user or client who made the request could not be authenticated. Verify the use
`403`	Forbidden. The user who made the request is not authorized to perform the action.
`404`	Not found. The requested object does not exist.

/environments/NAME/nodes

The `/environments/NAME/nodes` endpoint has the following methods: `GET`.

GET

The `GET` method is used to return a list of nodes in a given environment.

This method has no parameters.

Request

```
GET /organizations/NAME/environments/NAME/nodes
```

Response

The response is similar to:

```
{
  "blah": "https://api.opscode.com/org/org_name/nodes/_default",
  "boxer": "https://api.opscode.com/org/org_name/nodes/frontend",
  "blarrrrgh": "https://api.opscode.com/org/org_name/nodes/backend"
}
```

Response Codes

Response Code	Description
`200`	OK. The request was successful.
`401`	Unauthorized. The user or client who made the request could not be authenticated. Verify the use
`403`	Forbidden. The user who made the request is not authorized to perform the action.
`404`	Not found. The requested object does not exist.

/environments/NAME/recipes

The `/environments/NAME/recipes` endpoint has the following methods: `GET`.

GET

The `GET` method is used to return a list of recipes available to a given environment.

This method has no parameters.

Request

```
GET /organizations/NAME/environments/NAME/recipes
```

where the first instance of `NAME` is the name of the environment, and the second instance is the name of the recipe.

Response

The response is similar to:

```
[
  "ant",
  "apache2",
  "apache2::mod_auth_openid",
  "apache2::mod_authnz_ldap",
  "apt",
  "aws",
  "capistrano",
  "chef",
  "chef::bootstrap_client",
  "chef-client::config",
  "chef-client",
  ...
]
```

The list of recipes will be the default recipes for a given cookbook. If an environment has multiple versions of a cookbook that matches its constraints, only the recip

Response Codes

Response Code	Description
`200`	OK. The request was successful.
`401`	Unauthorized. The user or client who made the request could not be authenticated. Verify the use
`403`	Forbidden. The user who made the request is not authorized to perform the action.
`404`	Not found. The requested object does not exist.

/environments/NAME/roles/NAME

The `/environments/NAME/roles/NAME` endpoint has the following methods: `GET`.

GET

The `GET` method is used to return the `run_list` attribute of the role (when the name of the environment is `_default`) or to return `env_run_lists[environment_name`

{{< note >}}

The behavior of this endpoint is identical to `GET /roles/NAME/environments/NAME`; it is recommended (but not required) that `GET /roles/NAME/environments/NAME`

{{< /note >}}

This method has no parameters.

Request

```
GET /organizations/NAME/environments/NAME/roles/NAME
```

where the first instance of `NAME` is the name of the environment, and the second instance is the name of the role.

Response

The response is similar to:

```
{
  "run_list": [
    "recipe[recipe_name]",
    "role[role_name]",
    "recipe[recipe_name]",
    "role[role_name]",
    "recipe[recipe_name]",
    "role[role_name]"
  ]
}
```

Chef Infra Client will pick up the `_default` run-list if `env_run_list[environment_name]` is null or nonexistent.

Response Codes

Response Code	Description
`200`	OK. The request was successful.
`401`	Unauthorized. The user or client who made the request could not be authenticated. Verify the use
`403`	Forbidden. The user who made the request is not authorized to perform the action.
`404`	Not found. The requested object does not exist.

/groups

The `/groups` endpoint has the following methods: `GET` and `POST`.

GET

The `GET` method is used to get a list of groups on the Chef Infra Server for a single organization.

This method has no parameters.

Request

```
GET /organizations/NAME/groups
```

Response

The response is similar to:

```
{
  "33a5c28a8efe11e195005fsaes25400298d3f": "https://url/for/group1",
  "admins": "https://url/for/groups/admins",
  "billing-admins": "https://url/for/billing-admins",
  "clients": "https://url/for/clients",
  "developers": "https://url/for/developers",
  "users": "https://url/for/groups/users"
}
```

Response Codes

Response Code	Description
`200`	OK. The request was successful.
`401`	Unauthorized. The user or client who made the request could not be authenticated. Verify the use
`403`	Forbidden. The user who made the request is not authorized to perform the action.
`404`	Not found. The requested object does not exist.

POST

The `POST` method is used to create a group on the Chef Infra Server for a single organization.

Request

```
POST /organizations/NAME/groups
```

with a request body similar to:

```
{
  "name": "group1",
  "groupname": "group1",
  "orgname": "test",
  "actors": []
  "clients": ["mynode"],
  "groups": ["admins"],
  "users": ["betina"]
}
```

Response

The response is similar to:

```
{
  "uri": "https://chef.example/organizations/test/groups/group1",
}
```

Response Codes

Response Code	Description
`201`	OK. The group was created.
`401`	Unauthorized. The user or client who made the request could not be authenticated. Verify the use
`403`	Forbidden. The user who made the request is not authorized to perform the action.
`404`	Not found. The requested object does not exist.
`409`	Conflict. The requested group already exists.

/groups/GROUP_NAME

The `/groups/GROUP_NAME` endpoint has the following methods: `DELETE`, `GET` and `PUT`.

DELETE

The `DELETE` method is used to remove a group from a single organization.

This method has no parameters.

Request

```
DELETE /organizations/NAME/groups/GROUP_NAME
```

without a request body.

Response

The response is similar to:

```
{
}
```

Response Codes

Response Code	Description
`200`	OK. The group was deleted.
`401`	Unauthorized. The user or client who made the request could not be authenticated. Verify the use
`403`	Forbidden. The user who made the request is not authorized to perform the action.
`404`	Not found. The requested object does not exist.

GET

The `GET` method is used to get lists of users and other groups that belong to a group.

This method has no parameters.

Request

```
GET /organizations/NAME/groups/GROUP_NAME
```

Response

The response is similar to:

```
{
  "actors": [
    "pivotal",
    "grantmc"
  ],
  "users": [
    "pivotal",
    "grantmc"
  ],
  "clients": [

],
"groups": [
"000000000000ad94b5ddde157c070f0c"
],
"orgname": "inbetweens",
"name": "admins",
"groupname": "admins"
}
```

Response Codes

Response Code	Description
`200`	OK. The request was successful.
`401`	Unauthorized. The user or client who made the request could not be authenticated. Verify the use
`403`	Forbidden. The user who made the request is not authorized to perform the action.
`404`	Not found. The requested object does not exist.

PUT

The `PUT` method is used to update a group on a single organization.

This method has no parameters.

Request

```
PUT /organizations/NAME/groups/GROUP_NAME
```

with a request body similar to:

```
{
  "name": "group1",
  "groupname": "groupnew",
  "orgname": "test",
  "actors": []
  "clients": ["mynode","addme"],
  "groups": ["admins"],
  "users": ["betina"]
}
```

Response

The response is similar to:

```
{
  "name": "group1",
  "groupname": "groupnew",
  "orgname": "test",
  "actors": []
  "clients": ["mynode","addme"],
  "groups": ["admins"],
  "users": ["betina"]
}
```

Response Codes

Response Code	Description
`201`	OK. The group was updated.
`401`	Unauthorized. The user or client who made the request could not be authenticated. Verify the use
`403`	Forbidden. The user who made the request is not authorized to perform the action.
`404`	Not found. The requested object does not exist.

/nodes

{{% node %}}

The `/nodes` endpoint has the following methods: `GET` and `POST`.

GET

The `GET` method is used to return a hash of URIs for nodes on the Chef Infra Server.

This method has no parameters.

Request

```
GET /organizations/NAME/nodes
```

Response

The response is similar to:

```
{
  "latte": "https://localhost/nodes/latte"
}
```

Response Codes

Response Code	Description
`200`	OK. The request was successful.
`401`	Unauthorized. The user or client who made the request could not be authenticated. Verify the use
`403`	Forbidden. The user who made the request is not authorized to perform the action.

POST

The `POST` method is used to create a new node.

This method has no parameters.

Request

```
POST /organizations/NAME/nodes
```

with a request body similar to:

```
{
  "name": "latte",
  "chef_type": "node",
  "json_class": "Chef::Node",
  "attributes": {
    "hardware_type": "laptop"
  },
  "overrides": {},
  "defaults": {},
  "run_list": [ "recipe[unicorn]" ]
}
```

where `name` is the name of the node. Other attributes are optional. The order of the `run_list` attribute matters.

Response

The response is similar to:

```
{ "uri": "https://localhost/nodes/latte" }
```

Response Codes

Response Code	Description
`201`	Created. The object was created.
`400`	Bad request. The contents of the request are not formatted correctly.
`401`	Unauthorized. The user or client who made the request could not be authenticated. Verify the use
`403`	Forbidden. The user who made the request is not authorized to perform the action.
`409`	Conflict. The object already exists.
`413`	Request entity too large. A request may not be larger than 1000000 bytes.

/nodes/NAME

The `/nodes/NAME` endpoint has the following methods: `DELETE`, `GET`, and `PUT`.

DELETE

The `DELETE` method is used to delete a node.

This method has no parameters.

Request

```
DELETE /organizations/NAME/nodes/NAME
```

Response

The response will return the last known state of the node, similar to:

```
{
  "overrides": {},
  "name": "latte",
  "chef_type": "node",
  "json_class": "Chef::Node",
  "attributes": {
    "hardware_type": "laptop"
  },
  "run_list": [
    "recipe[apache2]"
  ],
  "defaults": {}
}
```

Response Codes

Response Code	Description
`200`	OK. The request was successful.
`401`	Unauthorized. The user or client who made the request could not be authenticated. Verify the use
`403`	Forbidden. The user who made the request is not authorized to perform the action.
`404`	Not found. The requested object does not exist.

GET

The `GET` method is used to return the details of a node as JSON.

This method has no parameters.

Request

```
GET /organizations/NAME/nodes/NAME
```

Response

The response is similar to:

```
{
  "name": "node_name",
  "chef_environment": "_default",
  "run_list": [
    "recipe[recipe_name]"
  ]
  "json_class": "Chef::Node",
  "chef_type": "node",
  "automatic": { ... },
  "normal": { "tags": [ ] },
  "default": { },
  "override": { }
}
```

Response Codes

Response Code	Description
`200`	OK. The request was successful.
`401`	Unauthorized. The user or client who made the request could not be authenticated. Verify the use
`403`	Forbidden. The user who made the request is not authorized to perform the action.
`404`	Not found. The requested object does not exist.

PUT

The `PUT` method is used to update a node.

This method has no parameters.

Request

```
PUT /organizations/NAME/nodes/NAME
```

with a request body similar to:

```
{
  "overrides": {},
  "name": "latte",
  "chef_type": "node",
  "json_class": "Chef::Node",
  "attributes": {
    "hardware_type": "laptop"
  },
  "run_list": [
    'recipe[cookbook_name::recipe_name],
    role[role_name]'
  ],
  "defaults": {}
}
```

Response

The response will return the updated node.

Response Codes

Response Code	Description
`200`	OK. The request was successful.
`401`	Unauthorized. The user or client who made the request could not be authenticated. Verify the use
`403`	Forbidden. The user who made the request is not authorized to perform the action.
`404`	Not found. The requested object does not exist.
`413`	Request entity too large. A request may not be larger than 1000000 bytes.

/policies

The `/policies` endpoint has the following methods: `GET`.

GET

The `GET` method is used to get a list of policies (including policy revisions) from the Chef Infra Server.

This method has no parameters.

Request

```
GET /organizations/NAME/policies
```

Response

The response groups policies by name and revision and is similar to:

```
{
  "aar": {
    "uri": "https://chef.example/organizations/org1/policies/aar",
    "revisions": {
      "37f9b658cdd1d9319bac8920581723efcc2014304b5f3827ee0779e10ffbdcc9": {
      },
      "95040c199302c85c9ccf1bcc6746968b820b1fa25d92477ea2ec5386cd58b9c5": {
      },
      "d81e80ae9bb9778e8c4b7652d29b11d2111e763a840d0cadb34b46a8b2ca4347": {
      }
    }
  },
  "jenkins": {
    "uri": "https://chef.example/organizations/org1/policies/jenkins",
    "revisions": {
      "613f803bdd035d574df7fa6da525b38df45a74ca82b38b79655efed8a189e073": {
      },
      "6fe753184c8946052d3231bb4212116df28d89a3a5f7ae52832ad408419dd5eb": {
      },
      "cc1a0801e75df1d1ea5b0d2c71ba7d31c539423b81478f65e6388b9ee415ad87": {
      }
    }
  }
}
```

Response Codes

Response Code	Description
`200`	OK. The request was successful.
`403`	Forbidden. The user who made the request is not authorized to pe

/policy_groups

The `/policy_groups` endpoint has the following methods: `GET`.

Each node has a 1:many relationship with policy settings stored on the Chef Infra Server. This relationship is based on the policy group to which the node is associat

- A policy is typically named after the functional role ahost performs, such as "application server", "chat server", "load balancer", and so on
- A policy group defines a set of hosts in a deployed units, typically mapped to organizational requirements such as "dev", "test", "staging", and "production", bu

/principals/NAME

The `/principals/NAME` endpoint has the following methods: `GET`.

GET

The `GET` method is used to get a list of public keys for clients and users in order to ensure that enough information is present for authorized requests.

This method has no parameters.

Request

```
GET /organizations/NAME/principals/NAME
```

Response

For a user or client, the type value will vary. The response body returns an array of principals which allows for a client with the same name as a user. The response fo

```
{
  "Principals: [
    {
      "name": "normal_user",
      "type": "user",
      "public_key": "-----BEGIN PUBLIC KEY-----...",
      "authz_id": "eca5fdd45a8b4bacc04bbc6e37a340bes",
      "org_member":false
    }
  ]
}
```

Response Codes

Response Code	Description
`200`	OK. The request was successful.
`404`	Not found. The requested object does not exist.

/roles

{{% role %}}

The `/roles` endpoint has the following methods: `GET` and `POST`.

GET

The `GET` method is used to get a list of roles along with their associated URIs.

This method has no parameters.

Request

```
GET /organizations/NAME/roles
```

Response

The response is similar to:

```
{
  "webserver": "https://chef.example/organizations/org1/roles/webserver"
}
```

Response Codes

Response Code	Description
`200`	OK. The request was successful.
`401`	Unauthorized. The user or client who made the request could not be authenticated. Verify the use
`403`	Forbidden. The user who made the request is not authorized to perform the action.

POST

The `POST` method is used to create a new role on the Chef Infra Server.

This method has no parameters.

Request

```
POST /organizations/NAME/roles
```

with a request body similar to:

```
{
  "name": "webserver",
  "default_attributes": {},
  "description": "A webserver",
  "env_run_lists": {
    "testenv": {
      "recipe[pegasus]"
    }
  },
  "run_list": [
    "recipe[unicorn]",
    "recipe[apache2]"
  ],
  "override_attributes": {}
}
```

Response

The response is similar to:

```
{ "uri": "https://chef.example/organizations/org1/roles/webserver" }
```

Response Codes

Response Code	Description
`201`	OK. The request was successful.
`400`	Bad request. The contents of the request are not formatted correctly.
`401`	Unauthorized. The user or client who made the request could not be authenticated. Verify the use
`403`	Forbidden. The user who made the request is not authorized to perform the action.
`409`	Conflict. The object already exists.
`413`	Request entity too large. A request may not be larger than 1000000 bytes.

/roles/NAME

The `/roles/NAME` endpoint has the following methods: `GET`, `DELETE`, and `PUT`.

DELETE

The `DELETE` method is used to delete a role on the Chef Infra Server.

This method has no parameters.

Request

```
DELETE /organizations/NAME/roles/NAME
```

Response

The response is similar to:

```
{
  "name": "webserver",
  "chef_type": "role",
  "json_class": "Chef::Role",
  "default_attributes": {},
  "description": "A webserver",
  "env_run_lists": {
    "env1": {
      "recipe[foo1]"
    }
  },
  "run_list": [
    "recipe[apache2]"
  ],
  "override_attributes": {}
}
```

Response Codes

Response Code	Description
`200`	OK. The request was successful.
`401`	Unauthorized. The user or client who made the request could not be authenticated. Verify the use
`403`	Forbidden. The user who made the request is not authorized to perform the action.
`404`	Not found. The requested object does not exist.

GET

The `GET` method is used to return the details for a role as JSON.

This method has no parameters.

Request

```
GET /organizations/NAME/roles/NAME
```

Response

The response is similar to:

```
{
  "name": "webserver",
  "chef_type": "role",
  "json_class": "Chef::Role",
  "default_attributes": {},
  "description": "A webserver",
  "env_run_lists": {},
  "run_list": [
    "recipe[unicorn]",
    "recipe[apache2]"
  ],
  "override_attributes": {}
}
```

Response Codes

Response Code	Description
`200`	OK. The request was successful.
`401`	Unauthorized. The user or client who made the request could not be authenticated. Verify the use
`403`	Forbidden. The user who made the request is not authorized to perform the action.
`404`	Not found. The requested object does not exist.

PUT

The `PUT` method is used to update a role on the Chef Infra Server.

This method has no parameters.

Request

```
PUT /organizations/NAME/roles/NAME
```

with a request body similar to:

```
{
  "name": "webserver",
  "default_attributes": {},
  "description": "A webserver",
  "env_run_lists": {},
  "default_attributes": {},
  "run_list": [
    "recipe[apache2]"
  ],
  "override_attributes": {}
}
```

Response

The response will return the JSON for the updated role.

Response Codes

Response Code	Description
`200`	OK. The request was successful.
`401`	Unauthorized. The user or client who made the request could not be authenticated. Verify the use
`403`	Forbidden. The user who made the request is not authorized to perform the action.
`404`	Not found. The requested object does not exist.
`413`	Request entity too large. A request may not be larger than 1000000 bytes.

/roles/NAME/environments

The `/roles/NAME/environments` endpoint has the following method: `GET`.

GET

The `GET` method returns a list of the environments that have environment-specific run-lists in the given role as JSON data.

This method has no parameters.

Request

```
GET /organizations/NAME/roles/NAME/environments
```

Response

The response is similar to:

```
["_default","production","qa"]
```

Response Codes

Response Code	Description
`200`	OK. The request was successful.
`401`	Unauthorized. The user or client who made the request could not be authenticated. Verify the use
`403`	Forbidden. The user who made the request is not authorized to perform the action.
`404`	Not found. The requested object does not exist.

/roles/NAME/environments/NAME

The `/roles/NAME/environments/NAME` endpoint has the following method: `GET`.

GET

The `GET` method returns the environment-specific run-list (`env_run_lists[environment_name]`) for a role.

This method has no parameters.

Request

```
GET /organizations/NAME/roles/NAME/environments/NAME
```

where the first `NAME` is the name of the role and the second is the name of the environment.

Response

The response is similar to:

```
{"run_list":["recipe[foo]"]}
```

Response Codes

Response Code	Description
`200`	OK. The request was successful.
`401`	Unauthorized. The user or client who made the request could not be authenticated. Verify the use
`403`	Forbidden. The user who made the request is not authorized to perform the action.
`404`	Not found. The requested object does not exist.

/sandboxes

A sandbox is used to commit files so they only need to be updated one time, as opposed to every time a cookbook is uploaded. The `/sandboxes` endpoint has the f

POST

The `POST` method is used to create a new sandbox. This method accepts a list of checksums as input and returns the URLs against which to `PUT` files that need to b

This method has no parameters.

Request

```
POST /organizations/NAME/sandboxes
```

with a request body similar to:

```
{"checksums": {
  "385ea5490c86570c7de71070bce9384a":null,
  "f6f73175e979bd90af6184ec277f760c":null,
  "2e03dd7e5b2e6c8eab1cf41ac61396d5":null
  }
}
```

Response

The response is similar to:

```
{"uri":
 "https://api.opscode.com/organizations/testorg/sandboxes/eff7b6f8b3ef44c6867216662d5eeb5f",
 "checksums":
   {"385ea5490c86570c7de71070bce9384a":
     {"url":
      "https://s3.amazonaws.com/opscode-platform-production-data/organization-(...)",
       "needs_upload":true},
       "f6f73175e979bd90af6184ec277f760c"=>
     {"url":
       "https://s3.amazonaws.com/opscode-platform-production-data/organization-(...)",
       "needs_upload":true},
       "2e03dd7e5b2e6c8eab1cf41ac61396d5":
     {"url":
       "https://s3.amazonaws.com/opscode-platform-production-data/organization-(...)",
       "needs_upload":true}
   },
 "sandbox_id"=>"eff7b6f8b3ef44c6867216662d5eeb5f"
}
```

Response Codes

Response Code	Description
`200`	OK. The request was successful. A hash that maps each checksum to a hash that contains a boolean `needs_u`
`400`	Bad request. The object has already been committed or one (or more) of the objects were not properly uploade containing a key for each checksum.

Response Code	Description
`401`	Unauthorized. The user or client who made the request could not be authenticated. Verify the user/client name
`403`	Forbidden. The user who made the request is not authorized to perform the action.
`413`	Request entity too large. A request may not be larger than 1000000 bytes.

/sandboxes/ID

Each sandbox has a unique identifier. The `/sandboxes/ID` endpoint has the following methods: `PUT`.

PUT

The `PUT` method is used to commit files that are in a sandbox to their final location so that changes to cookbooks will not require re-uploading the same data.

This method has no parameters.

Request

```
PUT /organizations/NAME/sandboxes/ID
```

with a request body similar to:

```
{"is_completed":true}
```

Response

The response is similar to:

```
{
  "guid": guid,
  "name": guid,
  "checksums":
    {"385ea5490c86570c7de71070bce9384a":
    {"url":
      "https://s3.amazonaws.com/opscode-platform-production-data/organization-(...)",
      "needs_upload":true}
  },
  "create_time": <get an example of time format>,
  "is_completed": true
}
```

Response Codes

Response Code	Description
`200`	OK. The request was successful.
`400`	Bad request. The contents of the request are not formatted correctly.
`401`	Unauthorized. The user or client who made the request could not be authenticated. Verify the use
`403`	Forbidden. The user who made the request is not authorized to perform the action.
`404`	Not found. The requested object does not exist.
`413`	Request entity too large. A request may not be larger than 1000000 bytes.

/search

{{% search %}}

The `/search` endpoint allows nodes, roles, data bags, environments to be searched. This endpoint has the following methods: `GET`.

{{< note >}}

At the end of every Chef Infra Client run, the node object is saved to the Chef Infra Server. From the Chef Infra Server, each node object is then added to the Apach
to the search index every 60 seconds or per 1000 node objects, whichever occurs first.

{{< /note >}}

{{< note >}}

This endpoint does not have any ACL restrictions, which means it may be used by any user or client that is able to make the request to the Chef Infra Server.

{{< /note >}}

GET

The `GET` method is used to return a data structure that contains links to each available search index. By default, the `role`, `node`, `client`, and `data bag` indexes v
Infra Server). Search indexes may lag behind the most current data at any given time. If a situation occurs where data needs to be written and then immediately sear

This method has no parameters.

Request

```
GET /organizations/NAME/search
```

This method has no request body.

Response

The response is similar to:

```
{
  "node": "https://localhost/search/node",
  "role": "https://localhost/search/role",
  "client": "https://localhost/search/client",
  "users": "https://localhost/search/users"
}
```

Response Codes

Response Code	Description
`200`	OK. The request was successful.
`401`	Unauthorized. The user or client who made the request could not be authenticated. Verify the use
`403`	Forbidden. The user who made the request is not authorized to perform the action.

/search/INDEX

Use the `/search/INDEX` endpoint to access the search indexes on the Chef Infra Server. The `/search/INDEX` endpoint has the following methods: `GET` and `POST`.

{{% search_query_syntax %}}

GET

The `GET` method is used to return all of the data that matches the query in the `GET` request.

This method has the following parameters:

Parameter	Description
`q`	The search query used to identify a list of items on a Chef Infra Server. This option use subcommand.
`rows`	The number of rows to be returned.
`start`	The row at which return results begin.

Request

```
GET /organizations/NAME/search/INDEX
```

Response

The response contains the total number of rows that match the request and is similar to:

```
{
 "total": 1,
 "start": 0,
 "rows": [
    {
     "overrides": {"hardware_type": "laptop"},
     "name": "latte",
     "chef_type": "node",
     "json_class": "Chef::Node",
     "attributes": {"hardware_type": "laptop"},
     "run_list": ["recipe[unicorn]"],
     "defaults": {}
    }
  ]
}
```

Response Codes

Response Code	Description
`200`	OK. The request was successful.
`401`	Unauthorized. The user or client who made the request could not be authenticated. Verify the use
`403`	Forbidden. The user who made the request is not authorized to perform the action.
`404`	Not found. The requested object does not exist.

POST

A partial search query allows a search query to be made against specific attribute keys that are stored on the Chef Infra Server. A partial search query can search th object index and providing a query that can be matched to the relevant index. While a full search query will return an array of objects that match (each object contai the attributes that match. One primary benefit of using a partial search query is that it requires less memory and network bandwidth while Chef Infra Client processe

To create a partial search query, use the `search` method, and then specify the key paths for the attributes to be returned. Each key path should be specified as an a

```
search(:node, 'role:web',
  :filter_result => { 'name' => [ 'name' ],
             'ip'   => [ 'ipaddress' ],
             'kernel_version' => [ 'kernel', 'version' ]
           }
) do |result|
  puts result['name']
  puts result['ip']
  puts result['kernel_version']
end
```

In the previous example, two attributes will be extracted (on the Chef Infra Server) from any node that matches the search query. The result will be a simple hash witl

The `POST` method is used to return partial search results. For example, if a node has the following:

```
{
  'x' => 'foo',
  'kernel' => { 'a' => 1, 'foo' => 'bar', 'version' => [ 1, 2, 3 ] }
}
```

a partial search query can be used to return something like:

```
{ 'kernel_version' => [ 1, 2, 3 ] }
```

This method has the following parameters:

Parameter	Description
`q`	The search query used to identify a list of items on a Chef Infra Server. This option uses the
`rows`	The number of rows to be returned.
`start`	The row at which return results begin.

Request

```
POST /organizations/NAME/search
```

with a request body similar to:

```
{
  "name": [ "name" ],
  "ip": [ "ipaddress" ],
  "kernel_version": [ "kernel", "version" ]
}
```

Response

The response is similar to:

```
{
  "name": "latte",
  "ip": "123.4.5.6789",
  "kernel_version": {"linux": "1.2.3"}
}
```

Response Codes

Response Code	Description
`200`	OK. The request was successful.
`401`	Unauthorized. The user or client who made the request could not be authenticated. Verify the use
`403`	Forbidden. The user who made the request is not authorized to perform the action.
`413`	Request entity too large. A request may not be larger than 1000000 bytes.

/universe

Use the `/universe` endpoint to retrieve the known collection of cookbooks, and then use it with Berkshelf and Chef Supermarket.

The `/universe` endpoint has the following methods: `GET`.

GET

The `GET` method is used to retrieve the universe data.

This method has no parameters. This method has no request body.

Request

```
GET /universe
```

Response

The response will return a json hash, with the name of each cookbook as a top-level key. Each cookbook will list each version, along with its location information an

```
{
  "ffmpeg": {
    "0.1.0": {
      "location_path": "http://supermarket.chef.io/api/v1/cookbooks/ffmpeg/0.1.0/download",
      "location_type": "supermarket",
      "dependencies": {
        "git": ">= 0.0.0",
        "build-essential": ">= 0.0.0",
        "libvpx": "~> 0.1.1",
        "x264": "~> 0.1.1"
      }
    },
    "0.1.1": {
      "location_path": "http://supermarket.chef.io/api/v1/cookbooks/ffmpeg/0.1.1/download",
      "location_type": "supermarket",
      "dependencies": {
        "git": ">= 0.0.0",
        "build-essential": ">= 0.0.0",
        "libvpx": "~> 0.1.1",
        "x264": "~> 0.1.1"
      }
    }
  },
  "pssh": {
    "0.1.0": {
      "location_path": "http://supermarket.chef.io/api/v1/cookbooks/pssh.1.0/download",
      "location_type": "supermarket",
      "dependencies": {}
    }
  }
}
```

Response Code	Description
`200`	OK. The request was successful. One (or more) cookbooks and associated cook returned.

/updated_since

The `/updated_since` endpoint ensures that replica instances of the Chef Infra Server are able to synchronize with the primary Chef Infra Server. `/updated_since` w and the `/updated_since` endpoint is also deprecated. The expectation for almost all chef users is that use of the endpoint will return an http status of 404. The `/org`

{{< warning >}}

This update is available after Chef replication is installed on the Chef Infra Server.

{{< /warning >}}

GET

The `GET` method is used to return the details of an organization as JSON.

Request

```
GET /organizations/NAME/updated_since?seq=NUM
```

where `NUM` is the largest integer previously returned as an identifier.

Response

The response will return an array of paths for objects that have been created, updated, or deleted since `NUM`, similar to:

```
[
  {
    "action": "create",
    "id": 1,
    "path": "/roles/foo"
  },
  {
    "action": "create",
    "id": 2,
    "path": "/roles/foo2"
  },
  {
    "action": "create",
    "id": 3,
    "path": "/roles/foo3"
  },
  {
    "action": "update",
    "id": 4,
    "path": "/roles/foo3"
  }
]
```

Response Codes

Response Code	Description
`200`	OK. The request was successful.
`401`	Unauthorized. The user or client who made the request could not be authenticated. Verify the use
`403`	Forbidden. The user who made the request is not authorized to perform the action.
`404`	Not found. The requested object does not exist or the function is not implemented.

/users

A user may be associated with an organization.

The `/users` endpoint has the following methods: `GET` and `POST`.

GET

The `GET` method is used to return an array of usernames for users associated with an organization.

This method has no parameters.

Request

```
GET /organizations/NAME/users
```

This method has no request body.

Response

The response is similar to:

```
[
  { "user": { "username": "paperlatte" } }
]
```

Response Codes

Response Code	Description
`200`	OK. The request was successful.
`401`	Unauthorized. The user or client who made the request could not be authenticated. Verify the use
`403`	Forbidden. The user who made the request is not authorized to perform the action.

POST

The `POST` method is used to associate a user with an organization immediately. Superuser only.

This method has no parameters.

Request

```
POST /organizations/NAME/users
```

with a request body similar to:

```
{
  "username": "paperlatte",
}
```

where `username` is the name of the user to be associated.

Response

No response block is returned.

Response Codes

Response Code	Description
`201`	Created. The user was associated with the organization.
`400`	Bad request. The contents of the request are not formatted correctly.
`401`	Unauthorized. The user or client who made the request could not be authenticated. Verify the use
`403`	Forbidden. The user who made the request is not authorized to perform the action.
`409`	Conflict. The user is already associated.

/users/NAME

The `/users/NAME` endpoint has the following methods: `DELETE`, `GET`.

DELETE

The `DELETE` method is used to delete a user association with an organization.

This method has no parameters.

Request

```
DELETE /organizations/NAME/users/NAME
```

Response

The response will return the end state of the user, similar to:

```
{
  "username": "paperlatte"
  "email": "latte",
  "display_name": "Ms. Latte",
  "first_name": "Paper",
  "last_name": "Latte",
  "public_key": "-----BEGIN PUBLIC KEY----- ... "
}
```

Response Codes

Response Code	Description
`200`	OK. The request was successful. The user association was removed.
`401`	Unauthorized. The user or client who made the request could not be authenticated. Verify the use
`403`	Forbidden. The user who made the request is not authorized to perform the action.
`404`	Not found. The requested object does not exist.

GET

The `GET` method is used to return the details of a user as JSON.

This method has no parameters.

Request

```
GET /organizations/NAME/users/NAME
```

Response

The response is similar to:

```
{
  "username": "paperlatte"
  "email": "latte",
  "display_name": "Ms. Latte",
  "first_name": "Paper",
  "last_name": "Latte",
  "public_key": "-----BEGIN PUBLIC KEY----- ... "
}
```

Response Codes

Response Code	Description
`200`	OK. The request was successful.
`401`	Unauthorized. The user or client who made the request could not be authenticated. Verify the use
`403`	Forbidden. The user who made the request is not authorized to perform the action.
`404`	Not found. The requested object does not exist.

Examples

The following sections show examples of using the Chef Infra Server API.

Query for Users and Orgs

The following example shows how to query the Chef Infra Server API for a listing of organizations and users. The `/organizations` and `/users` endpoints may only b installation of the Chef Infra Server.

Run the following from a `.chef` directory that contains a `pivotal.rb` file:

```
require 'chef'
require 'chef/server_api'

Chef::Config.from_file(".chef/pivotal.rb")
rest = Chef::ServerAPI.new(Chef::Config[:chef_server_url])
orgs = rest.get("/organizations")


puts "\n=== Listing of organizations"
orgs.each do |org|
puts org
end


puts "\n=== Listing of Users"
users = rest.get("/users")
users.each do |user|
puts user
end
```

An example of a `.chef/pivotal.rb` file is shown below:

```
current_dir = File.dirname(__FILE__)
node_name "pivotal"
chef_server_url "https://192.0.2.0:443"
chef_server_root "https://192.0.2.0:443"
client_key "#{current_dir}/pivotal.pem"
```

{{< note >}}

The `pivotal.pem` file must exist in the specified location and the IP addresses must be correct for the Chef Infra Server.

{{< /note >}}