API Umbrella Documentation

Release 0.14.4

National Renewable Energy Laboratory

Jul 15, 2017

Getting Started

CHAPTER 1

What Is API Umbrella?

API Umbrella is an open source API management platform for exposing web service APIs. The basic goal of API Umbrella is to make life easier for both API creators and API consumers. How?

- **Make life easier for API creators:** Allow API creators to focus on building APIs.
  - **Standardize the boring stuff:** APIs can assume the boring stuff (access control, rate limiting, analytics, etc.) is already taken care if the API is being accessed, so common functionality doesn't need to be implemented in the API code.
  - **Easy to add:** API Umbrella acts as a layer above your APIs, so your API code doesn't need to be modified to take advantage of the features provided.
  - **Scalability:** Make it easier to scale your APIs.
- **Make life easier for API consumers:** Let API consumers easily explore and use your APIs.
  - **Unify disparate APIs:** Present separate APIs as a cohesive offering to API consumers. APIs running on different servers or written in different programming languages can be exposed at a single endpoint for the API consumer.
  - **Standardize access:** All your APIs are can be accessed using the same API key credentials.
  - **Standardize documentation:** All your APIs are documented in a single place and in a similar fashion.

Installation

Installing From Binary Packages

Debian 8 (Jessie)

```
$ sudo apt-key adv --keyserver hkp://keyserver.ubuntu.com:80 --recv-keys␣
↪379CE192D401AB61
$ echo "deb https://dl.bintray.com/nrel/api-umbrella-debian jessie main" | sudo tee /
↪etc/apt/sources.list.d/api-umbrella.list
```

```
$ sudo apt-get update
$ sudo apt-get install api-umbrella
```

Debian 7 (Wheezy)

```
$ sudo apt-key adv --keyserver hkp://keyserver.ubuntu.com:80 --recv-keys␣
↪379CE192D401AB61
$ echo "deb https://dl.bintray.com/nrel/api-umbrella-debian wheezy main" | sudo tee /
↪etc/apt/sources.list.d/api-umbrella.list
$ sudo apt-get update
$ sudo apt-get install api-umbrella
```

Enterprise Linux 7 (CentOS/RedHat/Oracle/Scientific Linux)

```
$ curl https://bintray.com/nrel/api-umbrella-el7/rpm | sudo tee /etc/yum.repos.d/api-
↪umbrella.repo
$ sudo yum install api-umbrella
```

Enterprise Linux 6 (CentOS/RedHat/Oracle/Scientific Linux)

```
$ curl https://bintray.com/nrel/api-umbrella-el6/rpm | sudo tee /etc/yum.repos.d/api-
↪umbrella.repo
$ sudo yum install api-umbrella
```

Ubuntu 16.04 (Xenial)

```
$ sudo apt-key adv --keyserver hkp://keyserver.ubuntu.com:80 --recv-keys␣
↪379CE192D401AB61
$ echo "deb https://dl.bintray.com/nrel/api-umbrella-ubuntu xenial main" | sudo tee /
↪etc/apt/sources.list.d/api-umbrella.list
$ sudo apt-get update
$ sudo apt-get install api-umbrella
```

Ubuntu 14.04 (Trusty)

```
$ sudo apt-key adv --keyserver hkp://keyserver.ubuntu.com:80 --recv-keys␣
↪379CE192D401AB61
$ echo "deb https://dl.bintray.com/nrel/api-umbrella-ubuntu trusty main" | sudo tee /
↪etc/apt/sources.list.d/api-umbrella.list
$ sudo apt-get update
$ sudo apt-get install api-umbrella
```

Ubuntu 12.04 (Precise)

```
$ sudo apt-key adv --keyserver hkp://keyserver.ubuntu.com:80 --recv-keys␣
↪379CE192D401AB61
$ echo "deb https://dl.bintray.com/nrel/api-umbrella-ubuntu precise main" | sudo tee /
↪etc/apt/sources.list.d/api-umbrella.list
$ sudo apt-get update
$ sudo apt-get install api-umbrella
```

Installing With Chef

If you use Chef for managing your servers, we provide a cookbook that can be used to install the binary packages and configure API Umbrella.

Running With Docker

In this simple example, custom API Umbrella configuration can be defined in the `config/api-umbrella.yml` file on host machine. This gets mounted as `/etc/api-umbrella/api-umbrella.yml` inside the container, which is the path for the configuration file the rest of the documentation will reference.

```
$ mkdir config && touch config/api-umbrella.yml
$ docker run -d --name=api-umbrella -p 80:80 -p 443:443 -v "$(pwd)/config":/etc/api-
↪umbrella nrel/api-umbrella
```

Installing From Source Code

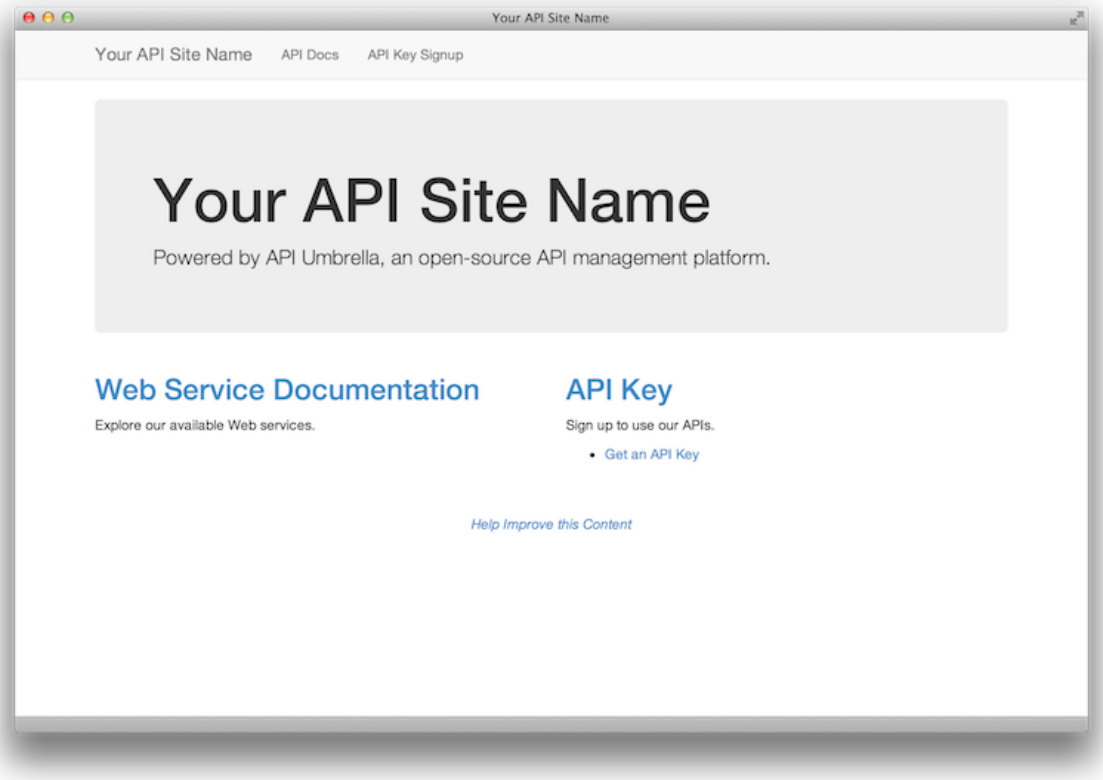
Installing from a binary package is recommended, if available (let us know if you'd like to see binary packages for other platforms). However, if a binary package is not available you can compile from source.

Setup

- Start API Umbrella:

```
$ sudo /etc/init.d/api-umbrella start
```

- Browse to your server's hostname. You should land on the default homepage:



Congrats! You're now up and running with API Umbrella. There are a variety of things you can do to start using the platform. Read on for a quick tutorial:

Quick Tutorial

Login to the web admin

A web admin is available to perform basic tasks:

`https://your-api-umbrella-host/admin/`

The very first time you access the admin, you'll be given a chance to create your first admin account.

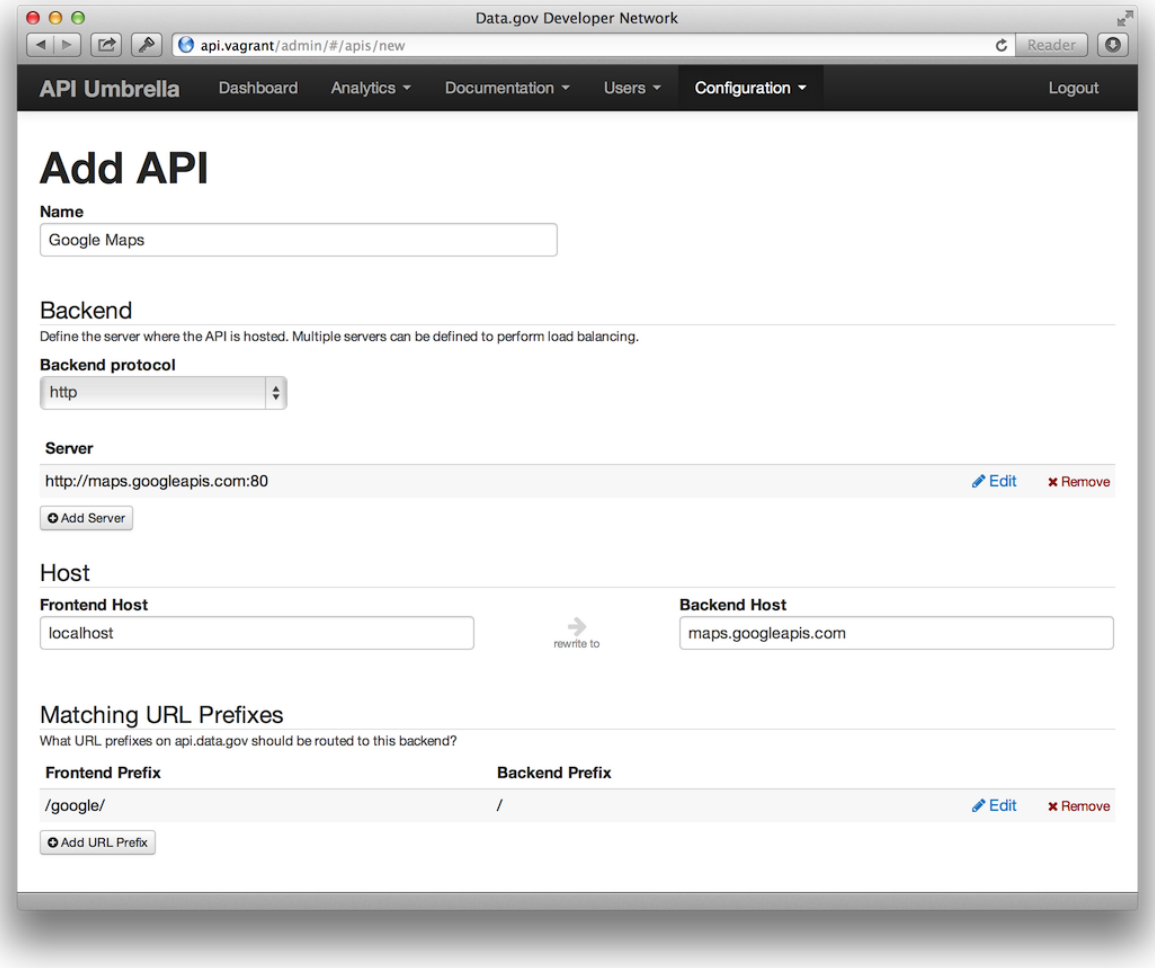
Add API Backends

Out of the box, API Umbrella doesn't know about any APIs. You must first configure the API backends that will be proxied to.

In this example, we'll proxy to Google's Geocoding API (but you'll more likely be proxying to your own web services).

Step 1: Login to the web admin and navigate to the "API Backends" section under the "Configuration" menu.

Step 2: Add a new backend:



Step 3: Navigate to the "Publish Changes" page under the "Configuration" menu and press the Publish button.

Google's API should now be available through the API Umbrella proxy.

Signup for an API key

On your local environment, visit the signup form:

`https://your-api-umbrella-host/signup/`

Signup to receive your own unique API key for your development environment.

Make an API request

Assuming you added the Google Geocoding example as an API backend, you should now be able to make a request to Google's Geocoding API proxied through your local API Umbrella instance:

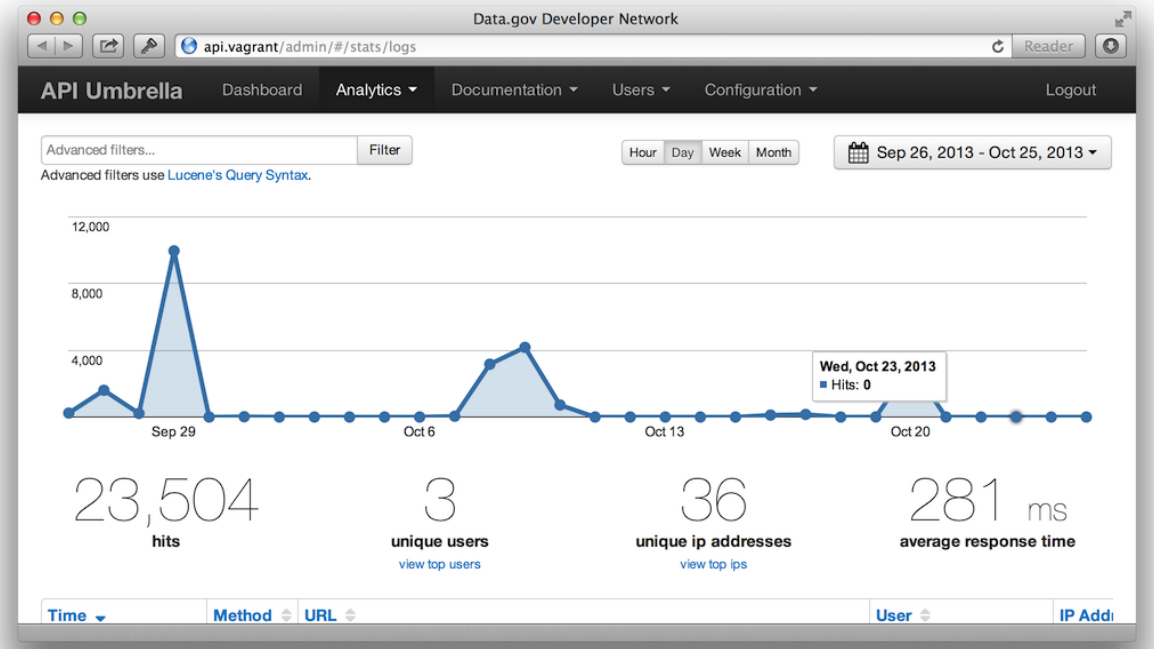
```
http://your-api-umbrella-host/google/maps/api/geocode/json?address=Golden,+CO&
↪sensor=false&api_key=**YOUR_KEY_HERE**
```

You can see how API Umbrella layers its authentication on top of existing APIs by making a request using an invalid key:

```
http://your-api-umbrella-host/google/maps/api/geocode/json?address=Golden,+CO&
↪sensor=false&api_key=INVALID_KEY
```

View Analytics

Login to the web admin. Navigate to the "Filter Logs" section under the "Analytics" menu. As you make API requests against your API Umbrella server, the requests should start to show up here (there may be a 30 second delay before the requests show up in the analytics).



Next Steps

API Backends

Caching

API Umbrella provides a standard HTTP caching layer in front of your APIs (using Apache Traffic Server). In order to utilize the cache, your API backend must set HTTP headers on the response. In addition to the standard `Cache-Control` or `Expires` HTTP headers, we also support the `Surrogate-Control` header.

Surrogate-Control

The `Surrogate-Control` header will only have an effect on the API Umbrella cache. This header will be stripped before the response is delivered publicly.

```
Surrogate-Control: max-age=(time in seconds)
```

Cache-Control: s-maxage

The `Cache-Control: s-maxage` header will be respected by the API Umbrella cache, as well as any other intermediate caches between us and the user.

```
Cache-Control: s-maxage=(time in seconds)
```

Cache-Control: max-age

The `Cache-Control: max-age` header will be respected by the API Umbrella cache, intermediate caching servers, and the user's client.

```
Cache-Control: max-age=(time in seconds)
```

Expires

The `Expires` header will be respected by the API Umbrella cache, intermediate caching servers, and the user's client.

```
Expires: (HTTP date)
```

HTTP Headers

After API Umbrella verifies an incoming request (a valid API key, below rate limits, etc), it will then proxy the incoming request to your API backend. The request your API backend receives will have additional HTTP headers added to the request. These headers can optionally be used to identify details about the requesting user.

X-Api-User-Id

A unique identifier for the requesting user (UUID format). This should be used if your API backend needs to uniquely identify the user making the request.

```
X-Api-User-Id: (UUID)
```

Example:

```
X-Api-User-Id: d44a13a0-926a-11e3-baa8-0800200c9a66
```

X-Api-Roles

If the user accessing the API has roles assigned to them, these will be present in the `X-Api-Roles` header as a comma-separated list of roles:

```
X-Api-Roles: (comma separated list)
```

Example:

```
X-Api-Roles: write_permissions,private_access
```

X-Forwarded-For

Used for identifying the original IP address of the client. See X-Forwarded-For for usage and details.

```
X-Forwarded-For: (comma separated list)
```

Example:

```
X-Forwarded-For: 203.0.113.54, 198.51.100.18
```

X-Forwarded-Proto

The original protocol of the client's request (either `http` or `https`). This can be used to determine how the client originally connected to the API regardless of what protocol is being used for API backend communication.

```
X-Forwarded-Proto: (http or https)
```

Example:

```
X-Forwarded-Proto: https
```

X-Forwarded-Port

The original port of the client's request (for example, `80` or `443`). This can be used to determine how the client originally connected to the API regardless of what port is being used for API backend communication.

```
X-Forwarded-Port: (number)
```

Example:

```
X-Forwarded-Port: 443
```

X-Api-Umbrella-Request-Id

A unique string identifier for each individual request. This can be used in log data to trace a specific request through multiple servers or proxy layers.

```
X-Api-Umbrella-Request-Id: (unique string identifier)
```

Example:

```
X-Api-Umbrella-Request-Id: aelqdj9lfoe7c2itheg0
```

X-Api-Key (Deprecated)

Currently the API passed in by the user is passed along to API backends in the `X-Api-Key` header. However, this header is deprecated and will be removed in the future. Instead, the `X-Api-User-Id` should be used if you need to uniquely identify the requesting user.

```
X-Api-Key: (api key)
```

Example:

```
X-Api-Key: 5WH3bgykjP9ihtrRl5ib9nQY5NzUGOixdXjBnx18
```

Role Restrictions

API Umbrella's "roles" feature can be used to restrict access to APIs so that only certain API keys may access certain APIs.

Adding Roles to API Keys

To grant specific API keys a role:

1. In the Admin, under Users > API Users, find the API key you want to add roles to.
2. Under Permissions > Roles, enter roles to assign to this API key.
    - You can name roles however you'd like.
    - Existing roles will auto-complete, but new roles can be created by entering a new name.

Enforcing Role Requirements With API Umbrella

If you'd like for API Umbrella to enforce role restrictions, then role requirements can be defined within the API Backend configuration:

1. In the Admin, under Configuration > API Backends, choose your API Backend to edit.
2. Under Global Request Settings > Required Roles, enter roles to require.
    - You can name roles however you'd like.
    - Existing roles will auto-complete, but new roles can be created by entering a new name.
    - If multiple roles are set, then the API key must have all of the roles.
3. Save changes to the API Backend and publish the configuration changes.

Once configured, then only API keys with the required roles will be able to access your API backend. If an API key lacks all of the required roles then API Umbrella will reject the request with a 403 Forbidden error and your API will never receive the request.

Sub-URL Role Requirements

The API Backend's "Sub-URL Request Settings" can be used to define more granular role requirements. For example, this could be used to require roles on just a single API URL path, or to only require roles for POST/PUT write access.

Enforcing Role Requirements Inside Your API

Instead of enforcing role requirements at the API Umbrella proxy layer, you can also utilize the role information in other ways within your API backend. If you have more complex authorization logic, then this may be easier to implement within your API's code.

On each request with a valid API key that's passed to your API backend, there's an `X-Api-Roles` HTTP header. This contains a comma-delimited list of all the roles assigned to the API key that's making the request. Your APIs can parse this HTTP header and use it to decide whether access should be permitted or denied.

API Users

Roles

API Umbrella's "roles" feature can be used to restrict access to APIs so that only certain API keys may access the API.

See the API Backends' Role Restrictions documentation for more detail.

Admin Accounts

Permissions

API Scopes are combined with the Admin Groups to create a granular permission system within the API Umbrella admin. This might be useful if you have multiple organizations or departments that should only have access to certain parts of the API Umbrella admin.

An API Scope defines a hostname and a path prefix. This determines the API backends and analytics that an admin is allowed to view. For example, an admin may be authorized to interact with `example.com/foo/*` apis, but not `example.com/bar/*` apis.

Next, you setup a (permissions) group, which defines the specific permissions admins can perform within API scopes. For example, you may want some admins to only be able to view analytics, while others should be able to also setup API backends.

As a quick example, say you set up an API Scope with a host of `example.com` and a path prefix of `/foo`. You then create a group that uses that scope and grants the `Analytics` and `API Backend Configuration - View & Manage` permissions. Then, you assign that group to a specific admin account.

Now, any admin that belongs to that group can log in and view analytics, but only for requests beginning with `example.com/foo/*`. They would not be able to view analytics for `example.com/bar/*`. Similarly, because they were granted the API Backend permission, that user could edit or create new API backends, but only as long as the API backend they're interacting with starts with `example.com/foo/*` for it's public URL. However, while this specific admin group could add and edit API backends, they couldn't actually publish the backend changes and make them live, since they were not granted the `API Backend Configuration - Publish` permission.

Analytics

Website Backends

By default, API Umbrella ships with a very basic public website. Any URL that does not match the routes defined by your API backends will get routed to your website backend. The default website provided by API Umbrella is intended to be customized or replaced. There are several different approaches to managing and hosting your website content:

- Using an *External Website Backend*
- Using the *Example Static Site*

External Website Backends

If you already have a website or content management system you'd like to use for managing your API website, you can point API Umbrella's website backend to wherever your website is hosted:

1. In the API Umbrella admin, under the "Configuration" menu pick "Website Backends" and then click "Add Website Backend"
2. Fill in the details for where your underlying website backend is hosted.
    - The "Frontend Host" field can be used if you'd like API Umbrella to handle multiple domain names and present different websites for each domain.
3. Save your website backend, and publish the changes under Configuration > Publish Changes.

Example Static Site

If you don't already have a preferred way for managing your websites, API Umbrella ships with a basic, example website. The default API Umbrella website content comes from the api-umbrella-static-site repository. This repository provides a Middleman site that can be forked and customized. As a static site, this site will compile to static HTML files and can be hosted in a variety of simple locations (GitHub Pages, an S3 Bucket, or on the API Umbrella servers).

Deployment

If you fork and customize the static site repository, you can then deploy it in a variety of ways. A few examples include:

- **External (GitHub Pages, S3 Bucket, etc):** You can deploy the resulting HTML files to these external locations, and then configure API Umbrella to point to these locations as you would any *external website backend*.
- **On the API Umbrella servers:** You'll need to configure your API Umbrella servers to allow for deployments, then you'll need to adjust the deploy configuration in `config/deploy/production.rb` to point to your servers, and then you should be able to deploy via `cap production deploy`.

Example Forks

Here are a couple examples of website repositories based on the api-umbrella-static-site repo, and deployed with GitHub Pages:

- api.data.gov
- developer.nrel.gov

REST API

Other Documentation

We are trying to consolidate all the API Umbrella documentation to this site, but in the meantime, some existing documentation can be found in a couple of other places:

- https://github.com/18F/api.data.gov/wiki/User-Manual:-Agencies
- https://github.com/NREL/api-umbrella/wiki

Admin Authentication

By default, API Umbrella's admin can be accessed with local admin accounts that can be created and managed without any further configuration.

The admin can also be configured to authenticate using external login providers (like Google, Facebook, or LDAP). These external login providers can be used in combination with local admin accounts, or local accounts can be disabled and external providers can be used exclusively.

General Configuration

Custom authentication settings can be defined in `/etc/api-umbrella/api-umbrella.yml`. The following example shows some general configuration options:

```
web:
  admin:
    auth_strategies:
      enabled:
        - local
        - github
        - google
    initial_superusers:
      - your.email@example.com
    username_is_email: true
```

- `web.admin.auth_strategies.enabled`: An array of authentication providers that should be enabled for logging into the admin (defaults to `local`). Available providers:
  - `local`
  - `cas`
  - `facebook`
  - `github`
  - `gitlab`
  - `google`
  - `ldap`
  - `max.gov`
- `web.admin.initial_superusers`: An array of superuser admin accounts that should be created on startup (defaults to none). Subsequent admin accounts can be created via the admin interface, so this setting is only needed for initial setup.

  When the local login provider is enabled (default), you will be given an opportunity to create an admin account on your first visit to the admin tool, so this option should not be necessary (unless you want to prevent the first visitor from being allowed to create an admin account).

  This option is primarily useful when the local login provider is disabled and you're exclusively using external login providers.
- `web.admin.username_is_email`: Whether or not the admin's username is also their email address (defaults to `true`). Setting this to `false` allows for a separate non-email based username to be assigned to admin accounts.

Local Login Provider

Example `/etc/api-umbrella/api-umbrella.yml` config:

```
web:
  admin:
    auth_strategies:
      enabled:
        - local
    password_length_min: 14
    password_length_max: 72
```

- `web.admin.password_length_min`: Minimum length of admin passwords (default `14`).
- `web.admin.password_length_max`: Maximum length of admin passwords (default `72`).

External Login Providers

CAS

Example `/etc/api-umbrella/api-umbrella.yml` config:

```
web:
  admin:
    auth_strategies:
      enabled:
        - cas
      cas:
        options:
          host: login.example.com
          login_url: /cas/login
          service_validate_url: /cas/serviceValidate
          logout_url: /cas/logout
          ssl: true
```

See omniauth-cas for further documentation.

Facebook

Example `/etc/api-umbrella/api-umbrella.yml` config:

```
web:
  admin:
    auth_strategies:
      enabled:
        - facebook
      facebook:
        client_id: "YOUR_CLIENT_ID_HERE"
        client_secret: "YOUR_CLIENT_SECRET_HERE"
```

To register your API Umbrella server with Facebook and get the `client_id` and `client_secret`:

1. Login to your Facebook developer account and add a new app.
2. Click **Add Product** in the left menu, and on the **Product Setup** screen, choose **Facebook Login**.

3. The **Valid OAuth redirect URIs** should be: `https://yourdomain.com/admins/auth/facebook/callback` (use the domain where API Umbrella is deployed)
4. Click **App Review** in the left menu, and flip the switch to make the app public.
5. Click **Settings** in the left menu and find your **App ID** and **App Secret**.
6. Add your `client_id` and `client_secret` to `/etc/api-umbrella/api-umbrella.yml`.
7. Reload API Umbrella (`sudo /etc/init.d/api-umbrella reload`).

See omniauth-facebook for further documentation.

GitHub

Example `/etc/api-umbrella/api-umbrella.yml` config:

```
web:
  admin:
    auth_strategies:
      enabled:
        - github
      github:
        client_id: "YOUR_CLIENT_ID_HERE"
        client_secret: "YOUR_CLIENT_SECRET_HERE"
```

To register your API Umbrella server with GitHub and get the `client_id` and `client_secret`:

1. Login to your GitHub account and create a new application.
2. The **Homepage URL** should be: `https://yourdomain.com` (use the domain where API Umbrella is deployed)
3. The **Authorization callback URL** should be: `https://yourdomain.com/admins/auth/github/callback`
4. Add your `client_id` and `client_secret` to `/etc/api-umbrella/api-umbrella.yml`.
5. Reload API Umbrella (`sudo /etc/init.d/api-umbrella reload`).

See omniauth-github for further documentation.

GitLab

Example `/etc/api-umbrella/api-umbrella.yml` config:

```
web:
  admin:
    auth_strategies:
      enabled:
        - gitlab
      gitlab:
        client_id: "YOUR_CLIENT_ID_HERE"
        client_secret: "YOUR_CLIENT_SECRET_HERE"
```

To register your API Umbrella server with GitLab and get the `client_id` and `client_secret`:

1. Login to your GitLab account and create a new application.
2. The **Redirect URI** should be: `https://yourdomain.com/admins/auth/gitlab/callback` (use the domain where API Umbrella is deployed)

3. The **Scopes** should be: `read_user`
4. Add your `client_id` and `client_secret` to `/etc/api-umbrella/api-umbrella.yml`.
5. Reload API Umbrella (`sudo /etc/init.d/api-umbrella reload`).

See omniauth-gitlab for further documentation.

Google

Example `/etc/api-umbrella/api-umbrella.yml` config:

```
web:
  admin:
    auth_strategies:
      enabled:
        - google
      google:
        client_id: "YOUR_CLIENT_ID_HERE"
        client_secret: "YOUR_CLIENT_SECRET_HERE"
```

To register your API Umbrella server with Google and get the `client_id` and `client_secret`:

1. Login to the Google API Console.
2. Create a new project for your API Umbrella site.
3. Navigate to **API Manager > Library** and enable the **Contacts API** and **Google+ API** APIs.
4. Navigate to **API Manager > Credentials**.
5. Under the **OAuth consent screen** tab, enter a **Product name shown to users**.
6. Under the **Credentials** tab, click the **Create cedentials** button and pick **OAuth Client ID**.
7. The **Application Type** should be: **Web application**.
8. The **Authorized JavaScript origins** should be: `https://yourdomain.com` (use the domain where API Umbrella is deployed)
9. The **Authorized redirect URIs** should be: `https://example.com/admins/auth/google_oauth2/callback`
10. Add your `client_id` and `client_secret` to the **api-umbrella.yml**.
11. Reload API Umbrella (`sudo /etc/init.d/api-umbrella reload`).

See omniauth-google-oauth2 for further documentation.

LDAP

Example `/etc/api-umbrella/api-umbrella.yml` config:

```
web:
  admin:
    username_is_email: false
    auth_strategies:
      enabled:
        - ldap
      ldap:
        options:
          title: Your Company
```

```
          host: ldap.example.com
          port: 389
          method: plain
          base: dc=example,dc=com
          uid: sAMAccountName
```

It may be useful to set `web.admin.username_is_email` to `false` if your LDAP account uses usernames (instead of email addresses) to authenticate.

See omniauth-ldap for further documentation.

MAX.gov

Example `/etc/api-umbrella/api-umbrella.yml` config:

```
web:
  admin:
    auth_strategies:
      enabled:
        - max.gov
```

If your website is authorized to use MAX.gov, no further configuration is necessary.

HTTPS Configuration

By default, API Umbrella requires HTTPS for a variety of endpoints. On initial installation, API Umbrella will use as self-signed certificate which won't be valid for production use. For production, you have two primary options:

- **SSL Termination:** If you're placing API Umbrella behind a load balancer in a multi-server setup, you can handle the SSL termination with that external load balancer.

  SSL termination should work without any further configuration assuming your external load balancer passes the appropriate `X-Forwarded-Proto` and `X-Forwarded-Port` headers to API Umbrella. If your load balancer does not support setting these headers, then see how you can override public ports.

- **SSL Certificate Installation:** You can configure API Umbrella with a valid SSL certificate, rather than the self-signed default one. To do so, install the certificates on your server, and then adjust the `/etc/api-umbrella/api-umbrella.yml` to point to these certificate files for your domain:

  ```
  hosts:
    - hostname: api.example.com
      default: true
      ssl_cert: /etc/ssl/your_cert.crt
      ssl_cert_key: /etc/ssl/your_cert.key
  ```

  `ssl_cert` should point to a valid certificate file in the format supported by nginx's `ssl_certificate`.

  `ssl_cert_key` should point to a valid private key file in the format supported by nginx's `ssl_certificate_key`.

SMTP Configuration

In order for end-users to receive email notifications, API Umbrella needs to be configured with SMTP settings.

Inside the `/etc/api-umbrella/api-umbrella.yml` config file, add SMTP settings under the *web.mailer.smtp_settings* key. The configuration under this key gets passed directly to Rails's **Action Mailer smtp_settings** configuration.

As a quick example, your configuration might look something like:

```
web:
  mailer:
    smtp_settings:
      address: smtp.whatever.com
      authentication: login
      user_name: example
      password: super_secure_pass
```

Refer to the Action Mailer docs `smtp_settings` section for all the available options.

Multi-Server Setup

A *basic install* will result in all of the API Umbrella services running on a single server. You may wish to scale your installation to multiple servers for redundancy or performance reasons.

Services

The first thing to understand with a multi-server installation are the individual services that can be run on each server. By default, all required services are run, but you can explicitly configure which services get run if you wish to split things off to different servers (for example, separating your database servers from your proxy servers).

To define which services get run, define the `services` configuration inside your `/etc/api-umbrella/api-umbrella.yml` configuration file:

```
services:
  - general_db
  - log_db
  - router
  - web
```

This configuration enables all the available services. To disable a service, remove its line from the configuration.

The services available are:

- `general_db`: The MongoDB database used for configuration, user information, and other miscellaneous data.
- `log_db`: The Elasticsearch database used for logging and analytics.
- `router`: The core reverse proxy and routing capabilities of API Umbrella.
- `web`: The web application providing API Umbrella's administration app and REST APIs.

Suggested Server Setups

In general, you'll need at least 3 servers in a multi-server setup since the database servers need an odd number of members for failover and voting purposes (see MongoDB Replica Set Strategies). Here are some possible server setups:

- 3 servers with all services running on all servers:

    - 3 servers with `router`, `web`, `general_db`, and `log_db` services enabled.
- 5 servers with the databases running on separate servers:
    - 2 servers with `router` and `web` services enabled.
    - 3 servers with `general_db` and `log_db` services enabled.
- 4 servers with the databases running on separate servers, and a MongoDB arbiter running on one of the proxy servers for voting purposes:
    - 1 server with `router` and `web` services enabled.
    - 1 server with `router`, `web`, `general_db` services enabled (but with MongoDB configured to be an arbiter for voting purposes only).
    - 2 servers with `general_db` and `log_db` services enabled.

Load Balancing

If you have multiple proxy or web servers running, you'll need to load balance between these multiple API Umbrella servers from an external load balancer. For a highly available setup, using something like an AWS ELB (or your hosting provider's equivalent) is probably the easiest approach. Alternatives involve setting up your own load balancer (nginx, HAProxy, etc).

Database Configuration

Bind Address

By default, the database processes bind to `127.0.0.1`, which means they will only accept connections from the same server. If you decide to run the database processes on separate servers, or you have multiple database servers, then you'll need to adjust the bind addresses to allow for communication between servers.

Warning: Elasticsearch offers no built-in security, and by default, passwords are not enabled on MongoDB. So it's important that you do not expose the servers to the public or unprotected networks.

Changing the bind addresses to `0.0.0.0` will allow for database communication between servers, but this setting is only appropriate if you have other firewall or network restrictions in place to prevent public access. Again, **be careful** not to expose your database servers to the internet.

For the Elasticsearch servers (any server with the `log_db` role):

```
elasticsearch:
  embedded_server_config:
    network:
      host: 127.0.0.1
```

For the MongoDB servers (any server with the `general_db` role):

```
mongodb:
  embedded_server_config:
    net:
      bindIp: 127.0.0.1
```

Multiple Servers

If you have multiple database servers, you'll need to adjust the `/etc/api-umbrella/api-umbrella.yml` configuration on all the servers to define the addresses of each database servers.

For the Elasticsearch servers (any server with the `log_db` role), define the server IPs:

```
elasticsearch:
  hosts:
    - http://10.0.0.1:14002
    - http://10.0.0.2:14002
    - http://10.0.0.3:14002
```

For the MongoDB servers (any server with the `general_db` role), define the server IPs and replica set name:

```
mongodb:
  url: "mongodb://10.0.0.1:14001,10.0.0.2:14001,10.0.0.3:14001/api_umbrella"
  embedded_server_config:
    replication:
      replSetName: api-umbrella
```

Note that for MongoDB, you'll still need to follow the normal procedure to deploy a replica set (for example, running `rs.initiate()`). In order to connect to MongoDB on the API Umbrella servers you can use this command: `/opt/api-umbrella/embedded/bin/mongo --port 14001`

HTTP/HTTPS Listen Ports

By default API Umbrella will startup and listen on the default HTTP and HTTPS ports (80 and 443). If you'd like to run API Umbrella on different ports, you can make changes to the `/etc/api-umbrella/api-umbrella.yml` config file:

```
http_port: 8080
https_port: 8443
```

Override Public Ports

If API Umbrella is placed behind a load balancer or other proxy, it should generally work without further configuration if the load balancer passes back the `X-Forwarded-Proto` and `X-Forwarded-Port` headers. These headers are commonly passed by other proxies by default, and it is the recommended approach to ensuring users see.

However, if your load balancer does not support sending back `X-Forwarded-Proto` and `X-Forwarded-Port` headers, and API Umbrella's internal ports differ from the public-facing ports, then you can explicitly override the public-facing port and protocol. The following configuration options can be defined in `/etc/api-umbrella/api-umbrella.yml`:

- `override_public_http_port`: Override the public port used when API Umbrella receives traffic on its `http_port` listener.
- `override_public_http_proto`: Override the public protocol (`http` or `https`) used when API Umbrella receives traffic on its `http_port` listener.
- `override_public_https_port`: Override the public port used when API Umbrella receives traffic on its `https_port` listener.
- `override_public_https_proto`: Override the public protocol (`http` or `https`) used when API Umbrella receives traffic on its `https_port` listener.

As an example, if you're terminating SSL outside of API Umbrella and sending all traffic to API Umbrella's HTTP port, then you could force API Umbrella into thinking the traffic to API Umbrella's HTTP port was originally received over HTTPS by overriding the public port and protocol for HTTP traffic:

```
override_public_http_port: 443
override_public_http_proto: https
```

However, note that in this case, API Umbrella has no way to distinguish between traffic that was originally HTTP or HTTPS (since they're both received on API Umbrella's HTTP port), so we're assuming the SSL terminator has already forced all traffic to HTTPS.

Log Files

Log files for API Umbrella are stored in `/var/log/api-umbrella/`. Inside that directory you'll find subdirectories for each process API Umbrella runs. Some of the more relevant log files are highlighted below:

- `/var/log/api-umbrella/nginx/access.log`: nginx access for all requests log
- `/var/log/api-umbrella/nginx/current`: nginx error log
- `/var/log/api-umbrella/web-puma/current`: Log file for the Rails web app (providing the admin and APIs)
- `/var/log/api-umbrella/trafficserver/access.blog`: Binary log file for the Traffic Server cache server (use `/opt/api-umbrella/embedded/bin/traffic_logcat` to view)

Database Configuration

Bind Address

By default, API Umbrella's bundled databases only accept connections from the same server. If you're running multiple servers, you'll need to adjust the bind address settings.

MongoDB Authentication

1. Create a user account for API Umbrella:

```
$ /opt/api-umbrella/embedded/bin/mongo --port 14001
> use api_umbrella
> db.createUser({
    user: "api_umbrella",
    pwd: "super_secret_password_here",
    roles: [
      { role: "readWrite", db: "api_umbrella" },
      { role: "dbAdmin", db: "api_umbrella" },
    ]
  })
> exit
```

2. Enable authorization and add the login details to the `mongodb.url` setting (using the Connection String URI Format) inside the `/etc/api-umbrella/api-umbrella.yml` config file:

```
mongodb:
  url: "mongodb://api_umbrella:super_secret_password_here@127.0.0.1:14001/api_
↪umbrella"
  embedded_server_config:
    security:
      authorization: enabled
```

3. Restart API Umbrella: `sudo /etc/init.d/api-umbrella restart`

External Database Usage

API Umbrella bundles the recommended database versions inside its package. Using other database versions is not supported, but should work. A few known notes about compatibility:

- Elasticsearch 1
  - API Umbrella can be used with an Elasticsearch 1 instance by setting the following config option in `/etc/api-umbrella/api-umbrella.yml`:

```
elasticsearch:
   api_version: 1
```

- Elasticsearch 5
  - API Umbrella is not yet compatible with Elasticsearch 5.

Analytics Storage Adapters

API Umbrella can store analytics data in different types of databases, depending on your performance needs and volume of metrics. The adapter can be picked by setting the `analytics.adapter` option inside the `/etc/api-umbrella/api-umbrella.yml` configuration file.

Elasticsearch

`analytics.adapter: elasticsearch`

- Currently the default analytics store.
- Suitable for small to medium amounts of historical analytics data. *(TODO: Provide more definitive guidance on what small/medium/large amounts are)*
- API Umbrella ships with with a default ElasticSearch database that can be used or any ElasticSearch 1.7+ cluster can be used.

Kylin

`analytics.adapter: kylin`

- Suitable for large amounts of historical analytics data. *(TODO: Provide more definitive guidance on what small/medium/large amounts are)*
- Requires a functional Hadoop environment compatible with Kylin 1.2 (Hortonworks Data Platform (HDP) 2.2 is recommended).

- Requires Kafka (can be enabled as part of HDP).
- Requires the optional `api-umbrella-hadoop-analytics` package to be installed on the analytics database server. *(TODO: Link to hadoop-analytics package downloads once built)*

PostgreSQL

```
analytics.adapter:  postgresql
```

TODO: The PostgreSQL adapter doesn't currently exist, but the idea is to leverage the same SQL framework built for Kylin.

- Suitable for small amounts of historical analytics data. *(TODO: Provide more definitive guidance on what small/medium/large amounts are)*

PostgreSQL: Columnar Storage

- Suitable for small to medium amounts of historical analytics data. *(TODO: Provide more definitive guidance on what small/medium/large amounts are)*

For better analytics performance with larger volumes of analytics data, you can continue to use the PostgreSQL adapter, but with a compatible column-based variant:

- cstore_fdw
- Amazon Redshift

API Key Usage

A user may pass their API key in one of several ways. In order of precedence:

HTTP Header

Pass the API key into the `X-Api-Key` header:

```
curl -H 'X-Api-Key: DEMO_KEY' 'http://example.com/api'
```

GET Query Param

Pass the API key into the `api_key` GET query string parameter:

```
curl 'http://example.com/api?api_key=DEMO_KEY'
```

Note: The GET query parameter may be used for non-GET requests (such as POST and PUT).

HTTP Basic Auth Username

As an alternative, pass the API key as the username (with an empty password) using HTTP basic authentication:

```
$ curl 'http://DEMO_KEY@example.com/api'
```

Rate Limits

Limits are placed on the number of API requests you may make using your API key. Rate limits may vary by service, but the defaults are:

- **Hourly Limit:** 1,000 requests per hour

For each API key, these limits are applied across all API requests. Exceeding these limits will lead to your API key being temporarily blocked from making further requests. The block will automatically be lifted by waiting an hour.

How Do I See My Current Usage?

Your can check your current rate limit and usage details by inspecting the `X-RateLimit-Limit` and `X-RateLimit-Remaining` HTTP headers that are returned on every API response. For example, if an API has the default hourly limit of 1,000 request, after making 2 requests, you will receive these HTTP headers in the response of the second request:

```
X-RateLimit-Limit: 1000
X-RateLimit-Remaining: 998
```

Understanding Rate Limit Time Periods

Hourly Limit

The hourly counters for your API key reset on a rolling basis.

Example: If you made 500 requests at 10:15AM and 500 requests at 10:25AM, your API key would become temporarily blocked. This temporary block of your API key would cease at 11:15AM, at which point you could make 500 requests. At 11:25AM, you could then make another 500 requests.

Rate Limit Error Response

If your API key exceeds the rate limits, you will receive a response with an HTTP status code of 429 (Too Many Requests).

Architecture

API Umbrella is a reverse proxy that sits between your API users and your APIs:

In More Detail

Components

Gatekeeper

Development Setup

The easiest way to get started with API Umbrella development is to use Vagrant to setup a local development environment.

Prerequisites

- 64bit CPU - the development VM requires an 64bit CPU on the host machine
- VirtualBox (version 4.3 or higher)
- Vagrant (version 1.6 or higher)
- ChefDK (version 0.10 or higher)
- NFS: For Mac OS X or Linux host machines only:
  - Mac OS X: Already installed and running
  - Ubuntu: `sudo apt-get install nfs-kernel-server nfs-common portmap`

Setup

After installing VirtualBox and Vagrant, follow these steps:

```
# Install the required Vagrant plugins
$ vagrant plugin install vagrant-berkshelf

# Get the code and spinup your development VM
$ git clone https://github.com/NREL/api-umbrella.git
$ cd api-umbrella
$ vagrant up # This step compiles API Umbrella from source, so the first time
             # make take 30-40 minutes.
```

Assuming all goes smoothly, you should be able to see the homepage at http://10.10.33.2/.

If you run into issues when running `vagrant up`, try running `vagrant provision` once to see if the error reoccurs. This will pickup with the setup process from the last failure point, which can sometimes help resolve temporary issues.

If you're still having any difficulties getting the Vagrant environment setup, then open an issue.

Directory Structure

A quick overview of some of the relevant directories for development:

- `src/api-umbrella/cli`: The actions behind the `api-umbrella` command line tool.
- `src/api-umbrella/proxy`: The custom reverse proxy where API requests are validated before being allowed to the underlying API backend.
- `src/api-umbrella/web-app`: Provides the admin tool and APIs.
- `src/api-umbrella/web-app/spec`: Tests for the admin tool and APIs.
- `test`: Proxy tests and integration tests for the entire API Umbrella stack.

Making Code Changes

This development VM runs the various components in "development" mode, which typically means any code changes you make will immediately be reflected. However, this does mean this development VM will run API Umbrella slower than in production.

While you can typically edit files and see your changes, for certain types of application changes, you may need to restart the server processes. There are two ways to restart things if needed:

```
# These commands must be executed *inside* your Vagrant VM:
$ vagrant ssh

# Quick: This should restart most server processes you'll need as a developer,
# but this doesn't restart everything:
$ sudo /etc/init.d/api-umbrella reload

# Slow: Restarts everything:
$ sudo /etc/init.d/api-umbrella restart
```

Writing and Running Tests

See the testing section for more information about writing and running tests.

Customizing Your VM

The following environment variables can be set prior to running `vagrant up` if you wish to tune the local VM (for example, to give it more or less memory, pick a different IP address, or use a different base box):

```
API_UMBRELLA_VAGRANT_BOX=nrel/CentOS-6.7-x86_64
API_UMBRELLA_VAGRANT_MEMORY=2048
API_UMBRELLA_VAGRANT_CORES=2
API_UMBRELLA_VAGRANT_IP=10.10.33.2
API_UMBRELLA_VAGRANT_NFS=true
```

Testing

Test Suite

API Umbrella's test suite uses Ruby's minitest. All tests are located in the `test` directory. Tests are separated into these areas:

- `test/admin_ui`: Browser-based tests for the `admin-ui` component using Capybara.
- `test/apis`: HTTP tests for the internal APIs provided by API Umbrella.
- `test/processes`: Testing the behavior of API Umbrella's server processes.
- `test/proxy`: Testing the behavior of API Umbrella's proxy features.
- `test/testing_sanity_checks`: Tests to sanity check certain behaviors of the overall test suite.

Running Tests

Assuming you have a Vagrant development environment, you can run all the tests with:

```
$ cd /vagrant
$ make test
```

Running Individual Tests

If you'd like to run individual tests, rather than all the tests, there are a few different ways to do that:

```
# Run individual files or tests within the web-app test suite:
$ cd /vagrant
$ ruby test/apis/v1/admins/test_create.rb
```

Deploying From Git

API Umbrella should be installed onto servers using the binary packages. However, if you want to deploy more recent updates from master (or your own forked changes), then newer versions of the app can be deployed on top of a package-based installation. Deployments are automated through Capistrano.

Prerequisites

In order to run the deployment scripts, your local computer (or wherever you're deploying from) must have:

- git
- rsync
- Ruby 1.9+
- Ruby Bundler

If you have trouble getting any of these setup locally, you can also run deployments from the development virtual machine, which includes these dependencies.

Initial Server Setup

SSH Key Setup

On each server you wish to deploy to, you must setup SSH keys so that you can deploy as the `api-umbrella-deploy` user (this user is automatically created as part of the package installation). These steps only need to be performed once per server.

- On your computer:
  - Ensure you have SSH keys: You must have SSH keys setup on your local computer (or wherever you're deploying from). If you do not have SSH keys, see steps 1 & 2 from GitHub's Generating SSH keys guide for instructions.
  - Copy your public key: Copy the contents of your public key (often at `~/.ssh/id_rsa.pub`). For more tips on copying, or alternative locations for your public key, see step 4 from GitHub's Generating SSH keys guide.
- On each server:
  - With your public SSH key in hand from your own computer, follow these steps on each server, replacing `YOUR_PUBLIC_KEY` as appropriate:

```
$ echo "YOUR_PUBLIC_KEY" | sudo tee --append /home/api-umbrella-deploy/.ssh/
↪authorized_keys
```

Install Build Dependencies

On each server you wish to deploy to, you must install the system packages needed for building dependencies (for example, make, gcc, etc). This can be automated through the `build/scripts/install_build_dependencies` shell script:

- On each server:

```
$ curl -OLJ https://github.com/NREL/api-umbrella/archive/master.tar.gz
$ tar -xvf api-umbrella-master.tar.gz
$ cd api-umbrella-master
$ sudo ./build/scripts/install_build_dependencies
```

Deploying

- One-time local setup:
  - Check out the api-umbrella repository from git:

    ```
    $ git clone https://github.com/NREL/api-umbrella.git
    ```

  - Install the deployment dependencies from inside the `deploy` directory:

    ```
    $ cd api-umbrella/deploy
    $ bundle install
    ```

  - Define your destination servers: Add a `.env` file inside the `api-umbrella/deploy` directory defining the servers to deploy to for the "staging" or "production" environments:

    ```
    API_UMBRELLA_STAGING_SERVERS="10.0.0.1,10.0.0.2"
    API_UMBRELLA_PRODUCTION_SERVERS="10.0.10.1,10.0.10.2"
    ```

    Servers can be defined using hostnames or IP address. Multiple servers can be comma-delimited. In this example there are two staging servers (`10.0.0.1` and `10.0.0.2`), and two production servers (`10.0.10.1` and `10.0.10.2`).

- Deploy to either the "staging" or "production" environments:

```
$ cd api-umbrella/deploy
$ bundle exec cap staging deploy
$ bundle exec cap production deploy
```

Building Binary Packages

Prerequisites

- git
- Docker

Supported Distributions

Currently we build 64bit binary packages for the following distributions:

- Debian 7 (Wheezy)
- Debian 8 (Jessie)
- Enterprise Linux 6 (CentOS/RedHat/Oracle/Scientific Linux)
- Enterprise Linux 7 (CentOS/RedHat/Oracle/Scientific Linux)
- Ubuntu 12.04 (Precise)
- Ubuntu 14.04 (Trusty)

Building Packages

To build packages for the current API Umbrella version for all distributions:

```
$ git clone https://github.com/NREL/api-umbrella.git
$ cd api-umbrella
$ make -C build/package -j4 docker_all # Adjust concurrency with -j flag as desired
```

Packages for each distribution will be created inside an isolated docker container, with the resulting packages being placed in the `build/package/work/current` directory.

Publishing Packages

To publish the new binary packages to our BinTray repositories (which provide yum and apt repos):

```
$ BINTRAY_USERNAME=username BINTRAY_API_KEY=api_key ./build/package/publish
```

Building Docker Images

Prerequisites

- git
- Docker

Building Images

To build packages for the current API Umbrella version:

```
$ git clone https://github.com/NREL/api-umbrella.git
$ cd api-umbrella/docker
$ docker build -t nrel/api-umbrella:INSERT_VERSION_HERE .
$ docker tag nrel/api-umbrella:INSERT_VERSION_HERE nrel/api-umbrella:latest
```

Pushing to Docker Hub

To publish the new images to our Docker Hub repository:

```
$ docker push nrel/api-umbrella:INSERT_VERSION_HERE
$ docker push nrel/api-umbrella:latest
```

Compiling From Source

Installing from a binary package is recommended, if available (let us know if you'd like to see binary packages for other platforms). However, if you'd like to compile from source, follow these instructions:

Prerequisites

- 64bit Linux distribution
  - It should be possible to run against other 64bit *nix operating systems, but our build script currently has some hard-coded assumptions to a 64bit linux environment. File an issue if you'd like to see other operating systems supported.
- Dependencies can automatically be installed for supported distributions by running the `./build/scripts/install_build_dependencies` script. For unsupported distributions, view the `./build/package_dependencies.sh` file for a list of required packages.

Compiling & Installing

```
$ curl -OLJ https://github.com/NREL/api-umbrella/archive/v0.14.4.tar.gz
$ tar -xvf api-umbrella-0.14.4.tar.gz
$ cd api-umbrella-0.14.4
$ sudo ./build/scripts/install_build_dependencies
$ ./configure
$ make
$ sudo make install
```

Release Process

Some basic instructions to follow when releasing a new, stable version of API Umbrella.

- Update the version number in `src/api-umbrella/version.txt`
  - Use semantic versioning.
- Update CHANGELOG.md with release notes.
- Update other references to the version number:
  - Documentation:
    - `docs/conf.py`
    - `docs/developer/compiling-from-source.md`
  - Website:
    - `website/source/index.html.erb`
    - `website/source/install.html.erb`
  - `Dockerfile`
- Build and publish new binary packages.
- Build and publish new docker container.
- Add a new GitHub Release (use the same release notes from the CHANGELOG).

Analytics Architecture

Overview

Analytics data is gathered on each request made to API Umbrella and logged to a database. The basic flow of how analytics data gets logged is:

```
[nginx] => [rsyslog] => [storage database]
```

To explain each step:

- nginx logs individual request data in JSON format to a local rsyslog server over a TCP socket (using lua-resty-logger-socket).
- rsyslog's role in the middle is for a couple of primary purposes:
  - It buffers the data locally so that if the analytics server is down or requests are coming in too quickly for the database to handle, the data can be queued.
  - It can transform the data and send it to multiple different endpoints.
- The storage database stores the raw analytics data for further querying or processing.

API Umbrella supports different analytics databases:

Elasticsearch

Suitable for small to medium amounts of historical analytics data. *(TODO: Provide more definitive guidance on what small/medium/large amounts are)*

Ingest

Data is logged directly to Elasticsearch from rsyslog:

```
[nginx] ====> [rsyslog] ====> [Elasticsearch]
        JSON            JSON
```

- rsyslog buffers and sends data to Elasticseach using the Elasticsearch Bulk API.
- rsyslog's omelasticsearch output module is used.

Querying

The analytic APIs in the web application directly query Elasticsearch:

```
[api-umbrella-web-app] => [Elasticsearch]
```

PostgreSQL

TODO: The PostgreSQL adapter doesn't currently exist, but the idea is to leverage the same SQL framework built for Kylin.

Suitable for small amounts of historical analytics data, or small to medium amounts of data with a columnar storage extension. *(TODO: Provide more definitive guidance on what small/medium/large amounts are)*

Ingest

Data is logged directly to PostgreSQL from rsyslog:

```
[nginx] ====> [rsyslog] ===> [PostgreSQL]
        JSON            SQL
```

- rsyslog buffers and sends data to PostgreSQL as individual inserts.
- rsyslog's ompgsql output module is used.
- If rsyslog supports batched transactions in the future, we should switch to that: rsyslog#895

Querying

The analytic APIs in the web application directly query PostgreSQL:

```
[api-umbrella-web-app] ===> [PostgreSQL]
                       SQL
```

PostgreSQL: Columnar Storage

For better analytics performance with larger volumes of analytics data, you can continue to use the PostgreSQL adapter, but with a compatible column-based variant:

- cstore_fdw
- Amazon Redshift

When these are used, the SQL table design and process remains the same, only the underlying table storage is changed for better analytic query performance.

Kylin

Suitable for large amounts of historical analytics data. *(TODO: Provide more definitive guidance on what small/medium/large amounts are)*

This is the most complicated setup, but it allows for vastly improved querying performance when dealing with large amounts of historical data. This is achieved by using Kylin to pre-compute common aggregate totals. By pre-computing common aggregations, less hardware is needed than would otherwise be needed to quickly answer analytics queries over large amounts of data. Under this approach, analytics data may not be immediately available for querying, since additional processing is required.

Ingest

During ingest, there are several concurrent processes that play a role:

```
[nginx] ====> [rsyslog] ====> [Kafka] ====> [Flume] ====> [HDFS - JSON (temp)]
        JSON            JSON           JSON           JSON
```

```
[HDFS - JSON (temp)] => [API Umbrella Live Processor] => [Hive - ORC]
```

```
[Hive - ORC] => [API Umbrella Kylin Refresher] => [Kylin]
```

- rsyslog buffers and sends JSON messages to Kafka using the omkafka output module.
  - Kafka is used as an intermediate step as a reliable way to get messages in order to Flume, but primarily Kafka is being used because that's what Kylin's future streaming feature will require (so it seemed worth getting in place now).
- Flume takes messages off the Kafka queue and appends them to a gzipped JSON file stored inside Hadoop (HDFS).
  - The JSON files are flushed to HDFS every 15 seconds, and new files are created for each minute.
  - The per-minute JSON files are partitioned by the request timestamp and not the timestamp of when Flume is processing the message. This means Flume could be writing to a file from previous minutes if it's catching up with a backlog of data.
  - Kafka's stronger in-order handling of messages should ensure that the per-minute JSON files are written in order, and skipping between minutes should not be likely (although possible if an nginx server's clock is severely skewed or an nginx server goes offline, but still has queued up messages that could be sent if it rejoins later).
  - Flume plays a very similar role to rsyslog, but we use it because it has the best integration with the Hadoop ecosystem and writing to HDFS (I ran into multiple issues with rsyslog's native omhdfs and omhttpfs modules).
- The API Umbrella Live Processor task determines when a per-minute JSON file hasn't been touched in more than 1 minute, and then copies the data to the ORC file for permanent storage and querying in the Hive table.
  - The live data should usually make it's way to the permanent ORC storage within 2-3 minutes.
  - The ORC data is partitioned by day.
  - The data is converted from JSON to ORC using a Hive SQL command. Each minute of data is appended as a new ORC file within the overall ORC daily partition (which Hive simply treats as a single daily partition within the overall logs table).
  - Since the data is only appended, the same minute cannot be processed twice, which is why we give a minute buffer after the JSON file has ceased writing activity to convert it to ORC.
  - The ORC file format gives much better compression and querying performance than storing everything in JSON.
  - If a new ORC file is created for a new day, the partition will be added to the Hive table.
  - At the end of each day, overwrite the daily ORC file with a new, compacted file from the original JSON data. Writing the full day at once provides better querying performance than the many per-minute ORC files. By basing this daily file on the original JSON data, it also alleviates any rare edge-cases where the per-minute appender missed data.
  - Automatically remove old JSON minute data once it's no longer needed.
- The API Umbrella Kylin Refresher task is responsible for triggering Kylin builds to updated the pre-aggregated data.
  - At the end of each day, after writing the compacted ORC file for the full day, we then trigger a Kylin build for the most recent day's data.

This setup is unfortunately complicated with several moving pieces. However, there are several things that could potentially simplify this setup quite a bit in the future:

- Kylin Streaming: This would eliminate our need to constantly refresh Kylin throughout the day, and reduce the amount of time it would take live data to become available in Kylin's pre-aggregated results. This feature available as a prototype in Kylin 1.5, but we're still on 1.2, and we'll be waiting for this to stabilize and for more

documentation to come out. But basically, this should just act as another consumer of the Kafka queue, and then it would handle all the details of getting the data into Kylin.

- Flume Hive Sink: Even with Kylin streaming support, we will likely still need our own way to get the live data into the ORC-backed Hive table. Flume's Hive Sink offers a way to directly push data from Flume into a ORC table. Currently marked as a preview feature, I ran into memory growth and instability issues in my attempts to use it, but if this proves stable in the future, it could be a much easier path to populating the ORC tables directly and get rid of the need for temporary JSON (along with the edge conditions those bring).
- Kylin Hour Partitioning: A possible shorter-term improvement while waiting for Kylin streaming is the ability to refresh Kylin by hour partitions. This would be more efficient than our full day refreshes currently used. This is currently implemented in v1.5.0, but we first need to upgrade to 1.5 (we're holding back at 1.2 due to some other issues), and then KYLIN-1513 would be good to get fixed before.

Querying

The analytic APIs in the web application query Kylin or PrestoDB using SQL statements:

```
                            /==> [Kylin] ====> [HBase Aggregates]
                           /
[api-umbrella-web-app] ===>
                          SQL \
                               \==> [PrestoDB] => [Hive ORC Tables]
```

- Queries are attempted against Kylin first, since Kylin will provide the fastest answers from it's pre-computed aggregates.
  - Kylin will be unable to answer the query if the query involves dimensions that have not been pre-computed.
  - We've attempted to design the Kylin cubes with the dimensions that are involved in the most common queries. These are currently:
    * timestamp_tz_year
    * timestamp_tz_month
    * timestamp_tz_week
    * timestamp_tz_date
    * timestamp_tz_hour
    * request_url_host
    * request_url_path_level1
    * request_url_path_level2
    * request_url_path_level3
    * request_url_path_level4
    * request_url_path_level5
    * request_url_path_level6
    * user_id
    * request_ip
    * response_status
    * denied_reason

    * request_method
    * request_ip_country
    * request_ip_region
    * request_ip_city

  - We don't add all the columns/dimensions to the Kylin cubes, since each additional dimension exponentially increases the amount of data Kylin has to pre-compute (which can significantly increase processing time and storage).
  - Data must be processed into Kylin for it to be part of Kylin's results, so the results will typically lag 30-60 minutes behind live data.

- If Kylin fails for any reason (the query involves a column we haven't precomputed or Kylin is down), then we perform the same query against PrestoDB. This queries the underlying ORC tables stored in Hive (which is the same raw data Kylin bases its data cubes on).
  - PrestoDB is used to provide an ANSI SQL layer on top of Hive. This should provide better compatibility with the SQL queries we're sending to Kylin, since both Kylin and PrestoDB aim for ANSI SQL compatibility (unlike Hive, which uses a different SQL-like HiveQL).
  - PrestoDB also offers better performance (significant in some cases) for our SQL queries rather than querying Hive directly. PrestoDB has also been fairly optimized for querying ORC tables.
  - Queries hitting PrestoDB will be slower than Kylin-answered queries. Query times vary primary depending on how much data is being queried, but response times may range from 5 seconds to multiple minutes.
  - Data must be processed into the ORC-backed Hive table for it to be part of PrestoDB's results, so results will typically lag 2-3 minutes behind live data (and therefore differ from Kylin results).
  - *TODO: Currently there's a 60 second timeout on PrestoDB queries to prevent long-running queries from piling up and hogging resources. However, if we find that people need to run longer-running queries, we can adjust this. We'll also need to adjust the default 60 second proxy timeouts.*