Traffic Ops API Design
v. 1.0.0

ocket8888
ocket8888@apache.org

September 15, 2020
Content

Introduction
   Anatomy of a meeting

Motivations
   API Best Practices
   Data Model

Considerations and Learned Lessons
Introduction
Anatomy of a meeting

- Roughly 20 meetings
- Consistent attendance from about 4 people
- 1 hour duration
- Living document as a pull request
- Two merged "blueprints"
Motivations

There are two main concerns addressed in the API document/meetings:

- API Best Practices
  - Proper use of HTTP request methods
  - Clarification of status codes and their purposes
  - No more relationships-as-objects or "join table endpoints"
  - Difference between "alerts" and a response

- Data Model
  - Separation between database and API
  - Prefer name-based identification
  - Reduce harmful reflection
  - Reduce field "abuse"
There are two main concerns addressed in the API document/meetings:

**API Best Practices**
- Proper use of HTTP request methods
- Clarification of status codes and their purposes
- No more relationships-as-objects or "join table endpoints"
- Difference between "alerts" and a response

**Data Model**
- Separation between database and API
- Prefer name-based identification
- Reduce harmful reflection
- Reduce field "abuse"
Request methods should be indicative of the action being performed - i.e. no more endpoints like `GET /deliveryservices/xmlId/{{xmlID}}/sslkeys/delete`.

`GET` must never modify API objects in any way.

Recommend `PATCH` support - objects can be huge.

Example **PUT** request for updating a Delivery Service

```
{
    "xmlId": "demo1",
    "displayName": "demo1",
    "longDescription": "some extremely long string",
    "rawRemapText": "something that could break the CDN",
    "consistentHashQueryParams": ["xxx", "yyy", "zzz"]
}
```
Request methods should be indicative of the action being performed - i.e. no more endpoints like `GET /deliveryservices/xmlId/{{xmlID}}/sslkeys/delete`. `GET` must *never* modify API objects in any way. Recommend `PATCH` support - objects can be huge.

**Example PATCH request to update one field of a Delivery Service**

```json
{
    "consistentHashQueryParams": ["test", "quest"]
}
```
API Best Practices
Clarification of status codes and their purposes

▶ Use **403 Forbidden** for out-of-Tenant access, not **404 Not Found**
API Best Practices
Clarification of status codes and their purposes

- Use **403 Forbidden** for out-of-Tenant access, not **404 Not Found**
- Always use **500 Internal Server Error** to hide error details from clients
API Best Practices
Clarification of status codes and their purposes

- Use **403 Forbidden** for out-of-Tenant access, not **404 Not Found**
- Always use **500 Internal Server Error** to hide error details from clients
- **201 Created** for object creation
If an object contains a set, map, or list of other objects, they should be properties of that parent object - *not* manipulated by a separate endpoint.
If an object contains a set, map, or list of other objects, they should be properties of that parent object - *not* manipulated by a separate endpoint.

- `/servers/{{ID}}/deliveryservices`
If an object contains a set, map, or list of other objects, they should be properties of that parent object - *not* manipulated by a separate endpoint.

- `/servers/{{ID}}/deliveryservices`
- `/deliveryservices/{{ID}}/servers`
API Best Practices
No more relationships-as-objects or "join table endpoints"

If an object contains a set, map, or list of other objects, they should be properties of that parent object - *not* manipulated by a separate endpoint.

- `/servers/{ID}/deliveryservices`
- `/deliveryservices/{ID}/servers`
- `/deliveryservices/{xmlID}/servers`
API Best Practices
No more relationships-as-objects or "join table endpoints"

If an object contains a set, map, or list of other objects, they should be properties of that parent object - not manipulated by a separate endpoint.

- `/servers/{{ID}}/deliveryservices`
- `/deliveryservices/{{ID}}/servers`
- `/deliveryservices/{{xmlID}}/servers`
- `/deliveryservice_server/{{DSID}}/{{serverID}}`
API Best Practices
No more relationships-as-objects or "join table endpoints"

If an object contains a set, map, or list of other objects, they should be properties of that parent object - *not* manipulated by a separate endpoint.

- /servers/{{ID}}/deliveryservices
- /deliveryservices/{{ID}}/servers
- /deliveryservices/{{xmlID}}/servers
- /deliveryservice_server/{{DSID}}/{{serverID}}
- /deliveryserviceserver
API Best Practices
No more relationships-as-objects or "join table endpoints"

If an object contains a set, map, or list of other objects, they should be properties of that parent object - *not* manipulated by a separate endpoint.

- /servers/{{ID}}/deliveryservices
- /deliveryservices/{{ID}}/servers
- /deliveryservices/{{xmlID}}/servers
- /deliveryservice_server/{{DSID}}/{{serverID}}
- /deliveryserviceserver
- /deliveryservices/{{ID}}/servers/eligible
API v1 response to URL key generation endpoint

{
  "response": "Successfully generated and stored keys"
}

ocket8888 | Traffic Ops API Design
API Best Practices

Difference between "alerts" and a response

Response conforming to API guidelines

```json
{
   "alerts": [
      {
         "level": "success",
         "text": "Successfully generated and stored keys"
      }
   ],
   "response": {
      "key6": "JhGdpw5X9o8TqHfgezCm0bqb9SQPASWL",
      "key0": "D4AYzJ1AE2nYisA9MxMtY03TPDCHi9C",
      "key3": "W90YH1Gc_kYlYw5_I0LrkpV9J0zSIneI",
      "key2": "0qgEoDO7sUsugIQemZbwMwMt0tNCwB1sf",
      "key4": "aFJ2Gb7atmxVB8uv7T9S60aDml3ycpGf",
      "key1": "wnWNR1mCz104C7EFPtqHd0xUMQyNFhA",
      "key5": "SIwv3GOhWN7EE9wSwPFj18qE4M07sFxN"
   }
}
```
Exposing database tables directly inevitably leads to guidelines violations and an overall more fragile design. Endpoints should be designed to serve a purpose, not expose some data\(^1\).

\(^1\text{Although that could be the purpose.}\)
Data Model
Prefer name-based identification

- Names often need to be unique anyway
- Much easier to remember
- "N+1" query problem
Customizable types expose database structure - versioning nightmare

How to find out if a server is an Edge-tier cache server

1. Check its Type

^This exists by default in new installations/upgrades - but it can be deleted!
Customizable types expose database structure - versioning nightmare

How to find out if a server is an Edge-tier cache server

1. Check its Type
   1.1 Look up Type by ID

*aThis exists by default in new installations/upgrades - but it can be deleted!*
Customizable types expose database structure - versioning nightmare

How to find out if a server is an Edge-tier cache server

1. Check its Type
   1.1 Look up Type by ID
   1.2 Check if its Type’s Name is the type "EDGE"\(^a\)

\(^a\)This exists by default in new installations/upgrades - but it can be deleted!
Customizable types expose database structure - versioning nightmare

How to find out if a server is an Edge-tier cache server

1. Check its Type
   1.1 Look up Type by ID
   1.2 Check if its Type’s Name is the type "EDGE"\(^a\)
   1.3 Check if its Type’s Name matches the pattern `^EDGE_.*`

\(^a\)This exists by default in new installations/upgrades - but it can be deleted!
Customizable types expose database structure - versioning nightmare

### How to find out if a server is an Edge-tier cache server

1. **Check its Type**  
   1.1 Look up Type by ID  
   1.2 Check if its Type’s Name is the type "EDGE"\(^a\)  
   1.3 Check if its Type’s Name matches the pattern ^EDGE_.*  

2. **Check its Profile**

\(^a\)This exists by default in new installations/upgrades - but it can be deleted!
Data Model
Reduce harmful reflection

Customizable types expose database structure - versioning nightmare

How to find out if a server is an Edge-tier cache server

1. Check its Type
   1.1 Look up Type by ID
   1.2 Check if its Type’s Name is the type "EDGE"\(^a\)
   1.3 Check if its Type’s Name matches the pattern \(^\text{EDGE}_\cdot.*\)

2. Check its Profile
   2.1 Look up its Profile by ID

\(^a\)This exists by default in new installations/upgrades - but it can be deleted!
Customizable types expose database structure - versioning nightmare

How to find out if a server is an Edge-tier cache server

1. Check its Type
   1.1 Look up Type by ID
   1.2 Check if its Type’s Name is the type "EDGE"\(^a\)
   1.3 Check if its Type’s Name matches the pattern ^EDGE_.*

2. Check its Profile
   2.1 Look up its Profile by ID
   2.2 Check its Profile’s Type (same procedure as before - but looking for ATS_PROFILE)

\(^a\)This exists by default in new installations/upgrades - but it can be deleted!
Customizable types expose database structure - versioning nightmare

How to find out if a server is an Edge-tier cache server

1. Check its Type
   1.1 Look up Type by ID
   1.2 Check if its Type’s Name is the type "EDGE"\(^a\)
   1.3 Check if its Type’s Name matches the pattern \(^\text{EDGE}_\cdot\).*

2. Check its Profile
   2.1 Look up its Profile by ID
   2.2 Check its Profile’s Type (same procedure as before - but looking for \text{ATS_PROFILE})
   2.3 Check if its Profile’s Name matches the pattern \(^\text{EDGE}_\cdot\).*

\(^a\)This exists by default in new installations/upgrades - but it can be deleted!
"description" fields regularly house business logic parsed by third-party tools
Considerations and Learned Lessons

Breaking changes

Things you might not think are breaking can be breaking
Things you might not think are breaking can be breaking

- **201 Created** considered a failure
Considerations and Learned Lessons
Satisfy use cases

It’s not enough to prevent bad behavior - why it was being done must be considered
Considerations and Learned Lessons
Satisfy use cases

It’s not enough to prevent bad behavior - why it was being done must be considered

▶ Tags could alleviate the need for reflection and can replace some field abuse.
Considerations and Learned Lessons
Satisfy use cases

It’s not enough to prevent bad behavior - why it was being done must be considered

▶ Tags could alleviate the need for reflection and can replace some field abuse.
▶ Profiles express powerful configuration options
Considerations and Learned Lessons
Introduce changes in small chunks

Changes to the API must be small enough to digest - needs to be broken up into the smallest actionable chunks possible
Considerations and Learned Lessons
Introduce changes in small chunks

Changes to the API must be small enough to digest - needs to be broken up into the smallest actionable chunks possible

Breaking up servers into various types

- Cache servers
- Database servers
- Infrastructure servers
- Traffic Monitors
- Traffic Ops servers
- Traffic Portals
- Traffic Routers
- Traffic Stats servers
- Traffic Vaults
Considerations and Learned Lessons
Introduce changes in small chunks

Changes to the API must be small enough to digest - needs to be broken up into the smallest actionable chunks possible

Changing Delivery Services

- Change "Active" from boolean to enumerated string constants
- Rename XMLID
Knowing something is bad isn’t enough to get rid of it. Some things just can’t really be changed, because there’s no way to encompass all of their use-cases.
Knowing something is bad isn’t enough to get rid of it. Some things just can’t really be changed, because there’s no way to encompass all of their use-cases.

▶ ANY_MAP
Sometimes doing something better means adding a shim
Considerations and Learned Lessons
Shims happen

Sometimes doing something better means adding a shim

- The ALL CDN
Thank you for listening
Source code available on GitHub