

Welcome to

a//things.api



cloud  
elements

# meet the speakers

**Ramana Lashkar** / Managing Director, Cloud Elements India  
**Avinash** / Senior Staff Engineer



cloud  
elements

a//things.api

# purpose & agenda

## purpose

All Thing API meetup is for API practitioners, implementers and developers to come together to share their experience and learnings on defining, designing, building, launching, managing and consuming APIs based on REST/SOAP/GraphQL/JSON/XML technologies.

## agenda

- What is a REST API
- CRUD Operations
- Common Patterns & Best practices for defining the best REST API
- Authentication
- Errors

# identify yourself

## sample personas



### Dave, The Developer

Your raw, unedited API is targeted directly at Dave the Developer. He discovers new APIs through developer communities and builds his own integrations.



### Izzy, IT/Project Manager

Izzy knows the in's and out's of her company's apps, as each system that is purchased, Izzy works with to keep connected, sharing data seamlessly. Izzy is the technical integrator.

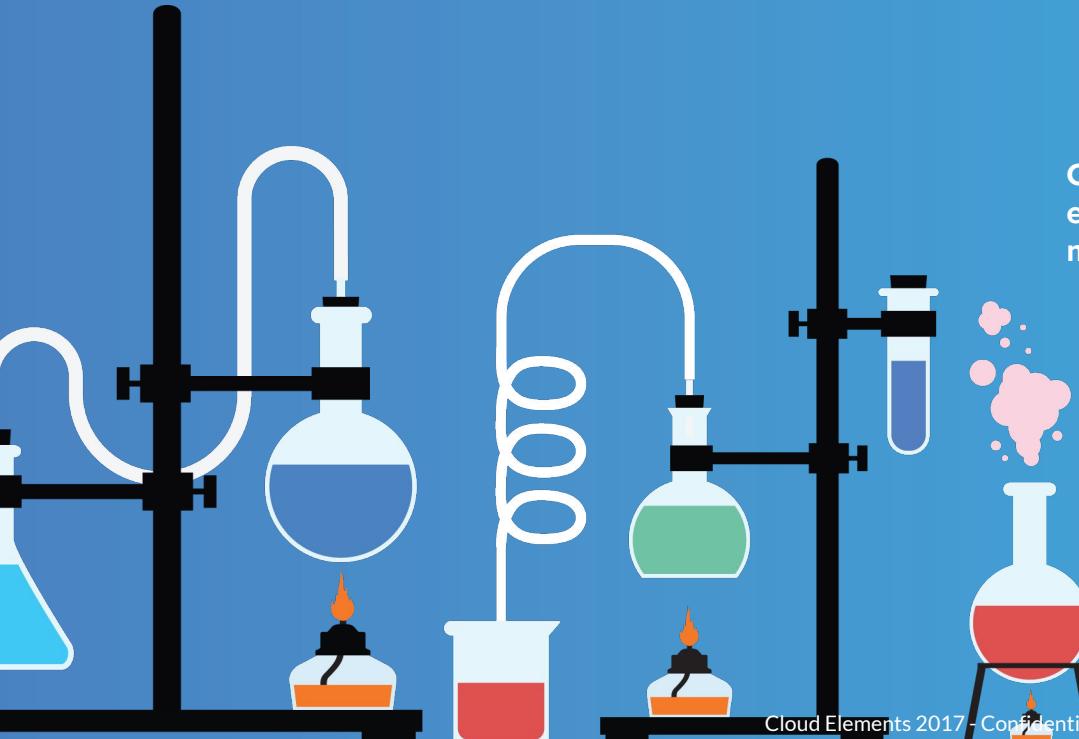


### Pete, Product Manager

As a product manager, Pete is responsible for connecting business needs with market demands, managing priorities, and meeting with customers.



cloud  
elements



# Who we are

Cloud Elements

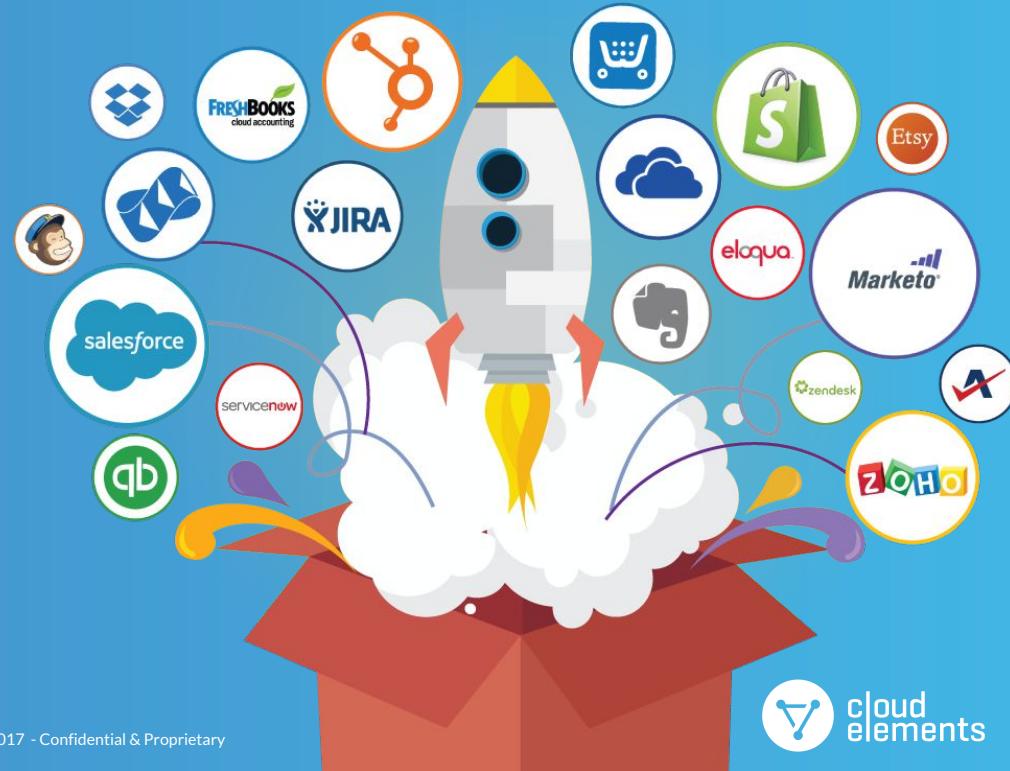
Cloud Elements is a cloud API integration platform that enables developers to publish, integrate, aggregate and manage all of their APIs through a unified platform.

# MAKING THE APIs YOU USE

## WORK TOGETHER

Embed integration right into your application.

Offer integrations that work right **out-of-the-box** for your customers.



A large blue iceberg is shown above the waterline, with a smaller whale swimming near its base. The water is a gradient from light blue at the top to dark blue at the bottom. In the upper right corner, there is an orange speech bubble containing the text "WRITING TO THE API".

WRITING TO  
THE API

Authenticate



Discover Custom Objects



Events Workflows



Map & Transform



Synchronize

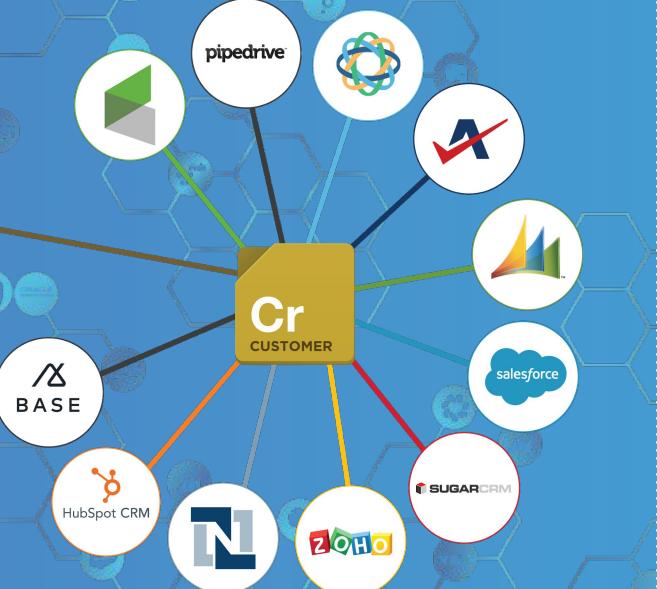
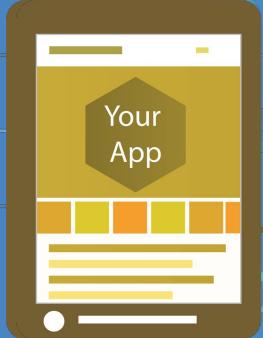


Log Bulk



# OUR HUB MODEL

ONE-TO-MANY INTEGRATIONS



## UNIFIED INTEGRATION WITH OUR **HUB APIs**

You integrate with our Hub APIs to connect with all services in that category

This reduces development cost, by delivering many integrations through a unified API

We handle the maintenance and roadmap for each element with minimal cost to you

## Cs CLOUD STORAGE



## Cr CRM



## Fi FINANCE



## Ma MARKETING



## Hd HELP DESK



## Ec ECOMMERCE



## Da DATABASE



## BETA Erp ERP



## Co COLLABORATION



## Hu HUMAN CAPITAL



## So SOCIAL



## Pa PAYMENTS



## Bi BILLING



## Ex EXPENSES



## Me MESSAGING



## Fs FIELD SERVICE



## Cf CONFERENCING



# ELEMENTS CATALOG

# Our Customers

[24]7

act-on

Autotask®

axway

Avalara

BrightTALK™

CONCUR

DEMANDBASE

freshdesk

fuze

FunnelWise™

GoToWebinar  
by CITRIX®

hootsuite

HEWLETT®  
PACKARD

IBM.

influitive  
advocate marketing experts

INSPIRATO®  
with American Express

KISSmetrics

leadspace

Lithium®

Microsoft

NINTEX

questback

ReadyTalk

sage

SAILTHRU

sprinklr

svb  
Silicon Valley Bank

SOCIAL123

sw!ftpage™

TIBC®

Tipalti

WESTERN UNION®

When I Work®

xerox



zendesk®

# What is an API

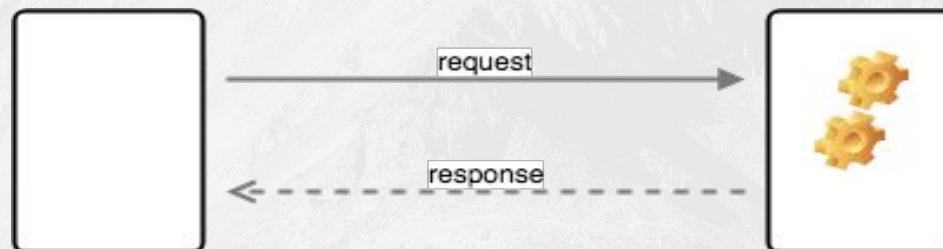
## What is an API ? How does it work ?

An API stands for an Application Programming Interface, which is a software intermediary that enables two or more applications to communicate with one another. APIs enable organizations to extract and share data in an accessible manner

Provides a predictable way to communicate between systems

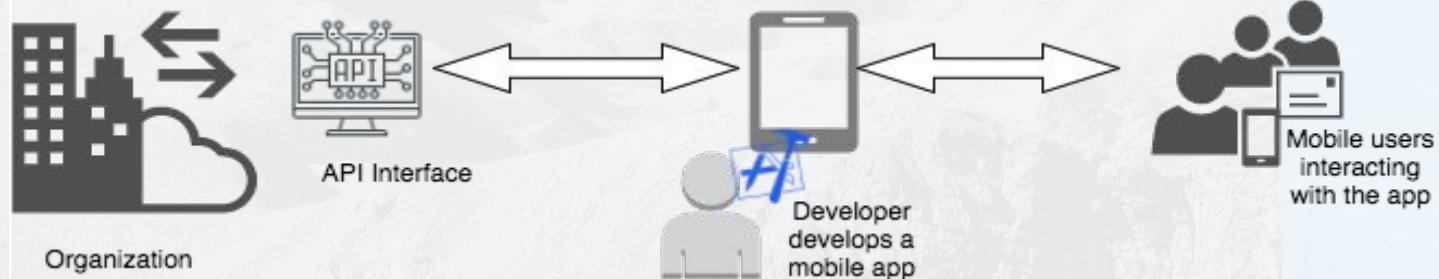
### How does an API work?

An API is a messenger that takes requests and tells the system what you want to do, then returns the response back to you.



# What is an API

## More Understanding of an API ?



# API Classification

## A Simple Classification of APIs

### Web service APIs

- SOAP
- RPC (XML, JSON)
- REST

### WebSocket APIs

### Library-based APIs

### Client Based APIs

- Java API
- Android API

### OS Functions and routines

- Access to file system
- Access to user interface

### Object remoting APIs

- CORBA
- .NET Remoting

### Hardware APIs

- drivers
- PCI buses

## Different Types of Web Service APIs

- **RPC**
  - RPCs are a form of interprocess communication (IPC), in that different processes have different address spaces
- **SOAP**
  - SOAP interfaces are method-based. Interface design is set of supported methods and data structures of each method
- **REST**
  - REST interfaces are resource-based. The most important aspect of the design is the URI structure that allows a consumer to navigate the object graph embodied by the API
- **GraphQL**
  - GraphQL is an open sourced API specification from Facebook that is touted as the next evolution from RESTful APIs.

# Representational State Transfer (REST)

## Representational State Transfer (REST)

### contacts

Show/Hide | List Operations | Expand Operations

GET /contacts

Search for contacts

POST /contacts

Create a contact

DELETE /contacts/{id}

Delete a contact

GET /contacts/{id}

Retrieve a contact

PATCH /contacts/{id}

Update a contact

### incidents

Show/Hide | List Operations | Expand Operations

GET /incidents

Search for incidents

POST /incidents

Create an incident

DELETE /incidents/{id}

Delete an incident

GET /incidents/{id}

Retrieve an incident

PATCH /incidents/{id}

Update an incident

GET /incidents/{id}/comments

Add a comment to an incident

# Representational State Transfer (REST)

## REpresentational State Transfer (REST)

- REST interfaces are Resource-based.
- Representational
- Six Constraints
  - Uniform Interface
  - Stateless
  - Cacheable
  - Client-Server
  - Layered System
  - Code on Demand (optional)

# Representational State Transfer (REST)

## REpresentational State Transfer (REST)

### RESOURCE BASED

- Things (resources) VS actions
- Nouns vs Verbs
- Identified by URLs
  - Example: `GET https://obscura.zendesk.com/api/v2/tickets`
  - `POST https://obscura.zendesk.com/api/v2/tickets`
- Separate from their representation(s)

### Representations

- Resource state transferred between client and server
- Typically JSON or XML
- Example :
  - Resource: person (ex: Ramana)
  - Service: contact information (GET)
  - Representation:
    - Name, address, phone
    - JSON or XML or YAML format
    - `{"name": "Ramana", "address": "Rent A Desk", "phone": "234-234-2345"}`
    - `<?xml version="1.0" encoding="UTF-8" ?><name>Ramana</name><address>Rent A Desk</address><phone>234-234-2345</phone>`

# Representational State Transfer (REST)

## REpresentational State Transfer (REST) Six Constraints

### Uniform Interface

- Interface between clients and servers
- It simplifies and decouples the architecture
- Fundamental to the RESTful design
- For us, this means...
  - HTTP Verbs (GET, POST, PATCH, PUT, DELETE)
  - URLs (resource names)
  - Http Response (status, body)

### Stateless

- Server contains no client state
- Each request contains enough context to process the message
- If any session state, it's held on the client

### Client-Server

- Always assume a disconnected system between client/server
- Separation of concerns
- Uniform interface is the link between the two

# Representational State Transfer (REST)

## **RE**presentational State Transfer (REST) Six Constraints

### **Cacheable**

- Server responses (representations) are cacheable
- Implicitly
- Explicitly (like max age, expires..etc)
- Negotiated

### **Layered System**

- Client can't assume direct connection to the server
- Software or Hardware intermediaries between client and servers
- This improves scalability

### **Code on Demand (Optional)**

- Servers can temporarily extend client
- Transfer logic to client
- Example
  - JavaScript, code snippet is sent to the client to execute

# REST

## HTTP VERBS

## HTTP Verbs

Most applications you have ever seen on the internet are CRUD (Create, Read, Update, Delete) applications and that the HTTP verbs match to these actions 1 to 1  
<http://tools.ietf.org/html/rfc2616>

### POST

Create

### GET

Read

### PUT

Update or Create

### PATCH

Update / Partially updating a resource

<http://tools.ietf.org/html/rfc5789>

### DELETE

As it says, DELETE

# POST

- POST is for creating a resources
- POST is neither safe nor idempotent. It is therefore recommended for non-idempotent resource requests.
- POST usually should accept a body

Example:

*POST /tickets*

```
{  
  "ticket": {  
    "subject": "My printer is on fire!",  
    "comment": {  
      "body": "The smoke is very colorful."  
    }  
  }  
}
```

Response

```
{  
  "ticket": {  
    "id": 35436,  
    "subject": "My printer is on fire!",  
    ...  
  }  
}
```

Response Status

Http Status Code: 200 OK or 201 CREATED

# GET

- GET is for getting the resources

Example: GET /tickets/35436

Response

```
{  
  "ticket": {  
    "id": 35436,  
    "subject": "My printer is on fire!",  
    "status": "open",  
    ...  
  },  
  "audit": {  
    "events": [...],  
    ...  
  }  
}
```

Get tickets GET /tickets

```
{  
  "tickets": [  
    {  
      "id": 35436,  
      "subject": "Help I need somebody!",  
      ...  
    },  
    {  
      "id": 20057623,  
      "subject": "Not just anybody!",  
      ...  
    }  
  ]  
}
```

Response Status

Http Status Code: 200 OK

# PUT

- PUT is for creating or replacing a resource
- PUT on an existing resource will replace the resource with the new details
- PUT usually should accept a body for payload and resource id in the PATH

Example: **PUT /tickets/35436**

```
{  
  "ticket": {  
    "subject": "My printer is on fire! updated",  
    "comment": {  
      "body": "The smoke is very colorful...now its all dark"  
    }  
  }  
}
```

**Response**

```
{  
  "ticket": {  
    "id": 35436,  
    "subject": "My printer is on fire! updated",  
    ...  
  }  
}
```

**Response Status**

Http Status Code: 200 OK or 201 if its CREATED

# PATCH

- PATCH is for partially updating a resource
- PATCH can have only the fields you want to modify and the ones that are not passed in gets unchanged

Example: PATCH /tickets/35436

```
{  
  "ticket": {  
    "subject": "My printer is on fire! Updated for Patch",  
  }  
}
```

Response

```
{  
  "ticket": {  
    "id": 35436,  
    "subject": "My printer is on fire! Updated for Patch",  
    ...  
  }  
}
```

Response Status

Http Status Code: 200 OK

# DELETE

- **DELETE** is for deleting a resource

Example: ***DELETE /tickets/35436***

**Response**

No Content

**Response Status**

Http Status Code: 204 No Content

# AUTHENTICATION

<p><b>Basic Auth:</b></p> <ul style="list-style-type: none"><li>- Widely used protocol for simple username/password</li><li>- Provides no confidentiality protection for the transmitted credentials.</li></ul>	<p><b>API Key:</b></p> <ul style="list-style-type: none"><li>- Doesn't require shared credentials</li><li>- Requires API to be accessed by a unique key comprised of numbers and letters</li><li>- Typically goes in the Authorization header in lieu of username and password</li></ul>
<p><b>SAML:</b></p> <ul style="list-style-type: none"><li>- XML-based open standard data format</li><li>- Popular with organizations using single sign-on, corporate applications and in many older legacy applications.</li></ul>	<p><b>OAuth &amp; OAuth 2:</b></p> <ul style="list-style-type: none"><li>- Authorize third-party access to their server resources without sharing their credentials</li><li>- Credential tokens are long lived, typically a year.</li><li>- OAuth 2 delegates security to the HTTPS protocol.</li><li>- OAuth 2 introduced the use of refresh tokens that allow authentications to expire</li><li>- OAuth 2 sometime requires pre- and post-hooks</li></ul>

# AUTHENTICATION EXAMPLE

GET /tickets/35436

Request Headers

Authorization: Basic gErypPlm4dOVgGRvA1ZzMH5MQ3nLo8bo==  
Content-Type: "application/json"  
Accept: "application/json"

Response

```
{  
  "ticket": {  
    "id": 35436,  
    "subject": "My printer is on fire!",  
    "status": "open",  
    ...  
  },  
  "audit": {  
    "events": [...],  
    ...  
  }  
}
```

Response Headers

X-Rate-Limit: 700

X-Rate-Limit-Remaining: 699

Response Status

Http Status Code: 200 OK

# ERRORS

## Standard Response Errors

2xx - received, understood, and accepted a request.

3xx - The caller must take further action in order to complete the request.

4xx - An error occurred in handling the request. The most common cause of this error is an invalid parameter.

5xx - received and accepted the request, but an error occurred in the service while handling it.

## Http Status Codes

200 success

201 created

202 accepted

204 no\_content

302 redirect

304 not\_modified

400 bad\_request

401 unauthorized

403 forbidden

404 not\_found

405 method\_not\_allowed

409 conflict

412 precondition\_failed

429 too\_many\_requests

500 internal\_server\_error

503 unavailable



cloud  
elements

a//things.api

# BEST PRACTICES

## REST URLs

POST /tickets

GET /tickets/{id}

PUT /tickets/{id}

PATCH /tickets/{id}

DELETE /tickets/{id}

GET /tickets (With search parameters and pagination)

## Sub resources

GET /tickets/{id}/comments

Sometimes you might end up having an action like below where it can't be represented as resource

POST /files/{id}/copy

## Response

- Use JSON than XML
- Consistency
- No Wrapper Objects in the response
- Include metadata in headers instead of response data
- Consistent Error Responses

# CLOSING THOUGHTS

- APIs **MISSION CRITICAL**
  - APIs are business Products
  - API should be part of your end-to end Business design
  - Consistency is the key factor
  - Version your APIs
- Some of the most **common errors** for include:
  - Authentication credentials change due to changes in passwords
  - API calls have changed and therefore failing
  - The service may be down completely or down for scheduled maintenance
- **Standardize error responses**, so that your support team can understand and react to errors more rapidly and efficiently vs. researching each endpoint.

## HELPFUL LINKS

<https://blog.cloud-elements.com>

<https://blog.cloud-elements.com/topic/developer>

<http://www.vinaysahni.com/best-practices-for-a-pragmatic-restful-api#restful>

<https://blog.cloud-elements.com/post-effyouthisistherighturl-restful-api-design>

<https://blog.cloud-elements.com/error-code-writing-good-api-status-codes>

<https://pages.apigee.com/rs/apigee/images/api-design-ebook-2012-03.pdf>



# Q&A

# We are Hiring!

<https://cloud-elements.com/careers/>



a//things.api