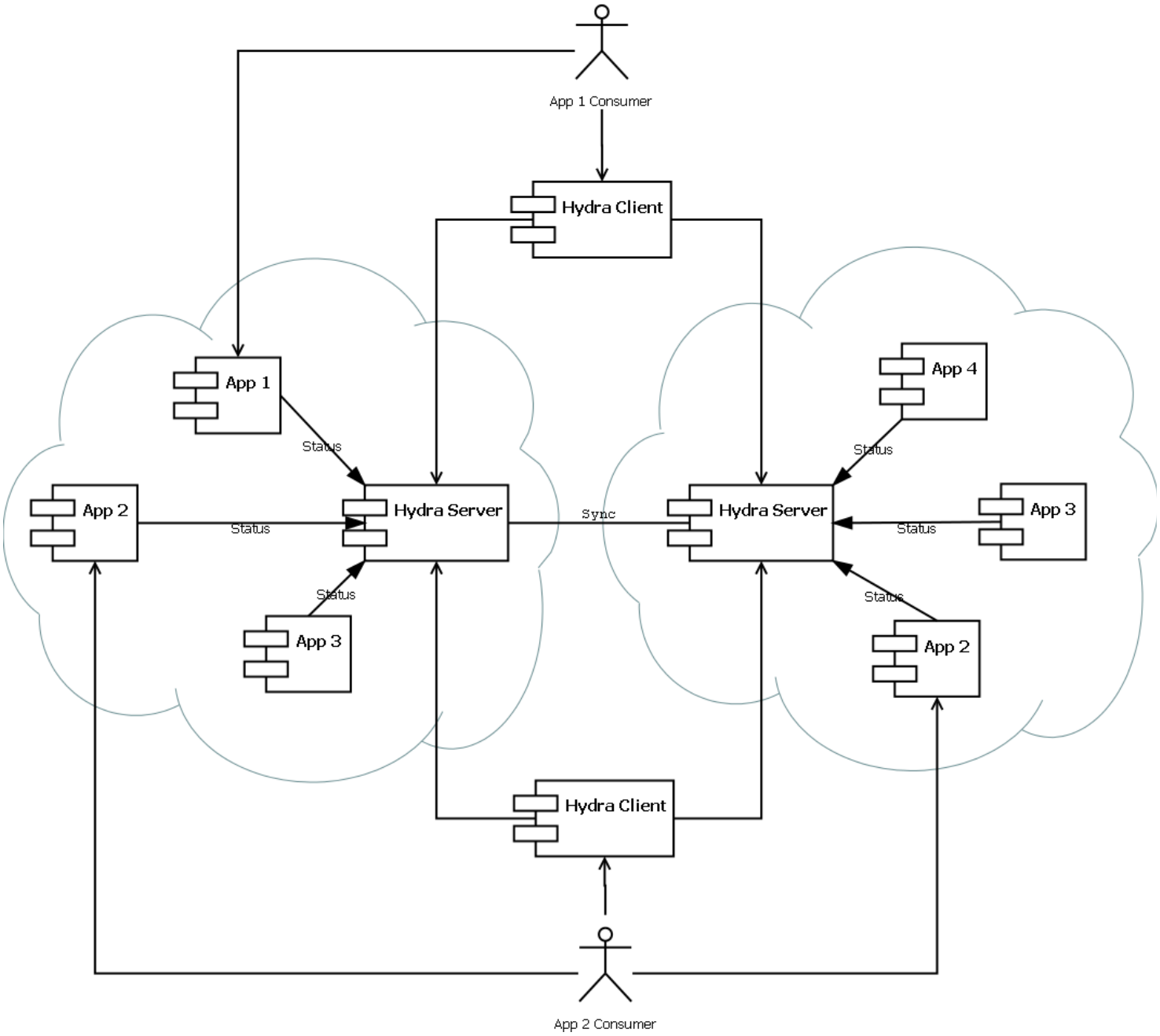
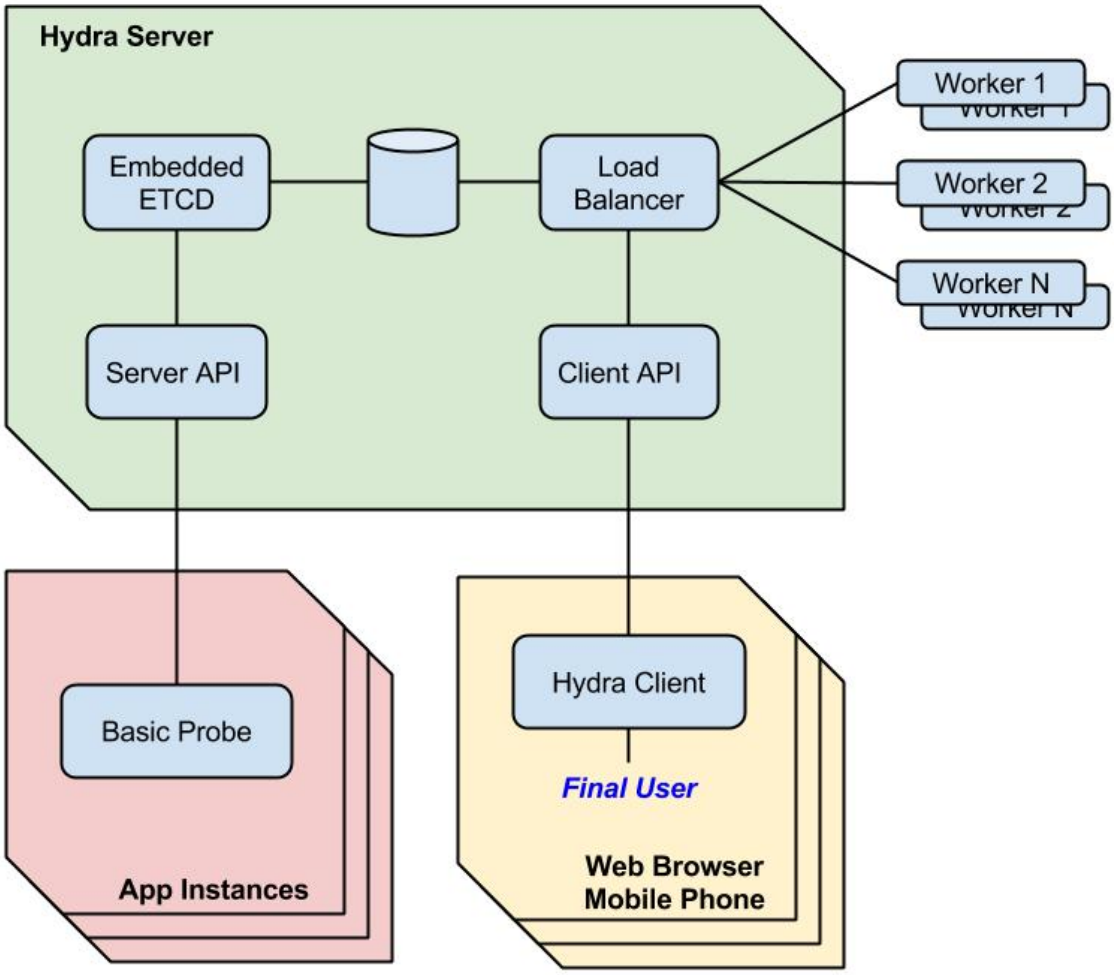
Hydra Concept

Multi-cloud application discovery, management and balancing.



Architecture



Implementation

Hydra Basic Probe

The Hydra Basic Probe is in charge of check and monitor one server and update the status information at one or several Hydra Servers using the restful server API.

The basic functionality is to notify to one Hydra Server when an application is Started, Stopping, or Removed. In addition, it will provide information about the server health status like CPU and memory usage and any useful information like the size of the server, TTL, etc.

All these information should be updated periodically. If not, the hydra server will assume that the servers are shutted down.

Hydra Server

Client API (Rest)

This module will expose a basic restful API for Hydra Client.

Operation	HTTP action	Description
Find a server/s for a given appID.	GET /apps/{appID}	Response example: [“http://mycompany.com/api”] Code 200 on success. Code 500 on error. *Maybe add some info about local balancing
Get the list of active hydra servers. Including itself.	GET /apps/hydra	Response example: [“http://hydra1.com:5000”, “http://hydra2.com:6000”] Code 200 on success. Code 500 on error.

Server API (Rest)

This module will expose a restful API for administration (Hydra Basic Probe and Monitor).

Operation	HTTP action	Description
Register or update an app	POST /apps/{appID}/Instances	Post data example: {"myserverName": {"cpuLoad": 91, "uri": "http://hydra-v3-demo-time-0.aws-ireland.innotechapp.com:8080", "connections": 0, "memLoad": 91, "state": 0, "cost": "3", "ttl": "20", "cloud": "aws-eu-west-1a"} } } Code 200 on success. Code 500 on error.

Get all apps	GET /apps/	Response example:
		[
		{
		"hydra": {
		"Balancers": {
		"0": {
		"simple": "OK",
		"worker": "RoundRobin"
		}
		},
		"Instances": {
		"ip-10-110-35-230": {
		"cloud": "aws-us-west-2b",
		"connections": "11.00",
		"cost": "9",
		"cpuLoad": "0.00",
		"demoMode": "unlocked",
		"memLoad": "14.30",
		"state": "0.00",
		"ttl": "20",
		"uri":
		"http://hydra-v3-demo-server-1.aws-oregon.innotechapp.com:7781"
		},
		"ip-10-112-20-17": {
		"cloud": "aws-eu-west-1a",

		"connections": "22.00", "cost": "3", "cpuLoad": "3.90", "demoMode": "unlocked", "memLoad": "17.50", "state": "0.00", "ttl": "20", "uri": "http://hydra-v3-demo-server-0.aws-ireland.innotechapp.com:7781" } } } }, ...] Code 200 on success. Code 500 on error.

Get one app	GET /apps/{appID}	Response example:
		{
		"hydra": {
		"Balancers": {
		"0": {
		"simple": "OK",
		"worker": "RoundRobin"
		}
		},
		"Instances": {
		"ip-10-110-35-230": {
		"cloud": "aws-us-west-2b",
		"connections": "11.00",
		"cost": "9",
		"cpuLoad": "0.00",
		"demoMode": "unlocked",
		"memLoad": "14.50",
		"state": "0.00",
		"ttl": "20",
		"uri":
		"http://hydra-v3-demo-server-1.aws-oregon.innotechapp.com:7781"
		},
		"ip-10-112-20-17": {
		"cloud": "aws-eu-west-1a",
		"connections": "20.00",
		"cost": "3",
		"cpuLoad": "4.00",
		"demoMode": "unlocked",

		"memLoad": "17.30", "state": "0.00", "ttl": "20", "uri": "http://hydra-v3-demo-server-0.aws-ireland.innotechapp.com:7781" } } } } Code 200 on success. Code 500 on error.

Balancing Chain

When you configure the master Hydra Server is mandatory to define the balancing chain that will be used for every application. For example:

```
[{
    "App1": {
        "Balancers": [
            {
                "worker": "MapAndSort",
                "mapAttr": "cloud",
                "mapSort": ["google", "amazon", "azure"]
            },
            {
                "worker": "SortByNumber",
                "sortAttr": "cpuLoad",
                "order": 1
            }
        ]
    }
}, {
    "App2": {
        "Balancers": [
            {
                "worker": "MapByLimit",
                "limitAttr": "limit",
                "limitValue": 50,
                "mapSort": "reverse"
            },
            {
                "worker": "RoundRobin"
                "simple": "OK"
            }
        ]
    }
}]
```

```
This configuration will define two possible apps (“App1” and “App2”). POST for differents apps will be
rejected from Hydra Server.

Balancing chain is defined in the object “Balancers” and they will be executed in every GET invocation
to the public API for that app.
```

Hydra Client

There are several Hydra clients for Java, Javascript Web, Javascript Node, etc. But the basic functionality is the same. We are going to explain here the basic Javascript client.

Hydra client is a js file that should be included in the web page or node project. It provides two functions:

function hydra.config([<server list>], options)

- [<server list>] - the initial servers we want to use to access Hydra.
- options - (optional) object containing the following fields:
    - hydraTimeOut - timeout for updating Hydra Servers. Minimum of 60 seconds
    - appTimeOut - timeout for updating an app server in the internal cache. Minimum 20 seconds
    - retryOnFail - retry timeout in case the hydra we are requesting an app or new Hydra servers fail to answer. Minimum 500ms

By default, the initial Hydra server will be the host serving the Hydra client file, making this function call optional on the browser, although it’s recommended to set up the servers.

function hydra.get(appID, nocache, callback)

This function will call to callback(error, [servers]) function with the url of the server that provides the given appID.

- appID - id of the application requested
- nocache - boolean, if set to true will ask the hydra server for the application servers ignoring the internal cache.
- callback(error, [servers]) - function callback that will receive the app server or an error in case the app does not exist

Internally, it will ask to the first Hydra server or use the internal cache in order to get the corresponding server url for the app and then it will call to callback function. If the application exist, the servers are sent back and served through the callback function (if the application exist, but there are no servers available, it will return an empty array). If the application does not exist, the callback will receive an error and the list will be set to null.

In case an Hydra server fails to answer (when requesting an app or new Hydra servers), the client will try again (based on the retryOnFail timeout) using the next server and moving the one that failed to the end of the list until one of the Hydra servers replies.

Example of use with pooling every minute

```
var app1url = []
var interval = setInterval(hydra.get(app1id, false, function(data) { app1url = data }, 60000)
…
$.ajax({
  url: app1url[0] ,
  sucess: {
     ….
  });
```

Example of use renewing server url only on error

```
var retry = 0
var NUM_RETRIES = 2
hydra(app1id, false, myfunction);
myfunction(app1url) {
  $.ajax({
    url: app1url[0] ,
    sucess: {
      retry = 0;
      ...
    },
    error: {
      if (retry<NUM_RETRIES ) {
        hydra.get(app1id, true, myfunction);
        retry++;
      }
    }
  });
}
```