ElasticHosts

Flexible servers in the cloud

Cloud infrastructure API standardization

Richard Davies

CEO, ElasticHosts Ltd

February 2009

Background

ElasticHosts

- Second European cloud infrastructure provider, public beta launched November 2008
- First public cloud based upon KVM, the native Linux virtualization platform
- API released[1] December 2008

Need for standardized API

- Stimulate ecosystem (e.g. CohesiveFT, RightScale) by enabling identical code to run against all clouds
- Counter customer concerns about vendor lock-in to a specific cloud

[1] See http://www.elastichosts.com/products/api

Ambitions for API standardization

Swift progress

- Amazon EC2 the de-facto standard today;
  new standard must swiftly gain momentum to survive
- Practical approach needed to rapidly agree on core functionality (e.g. starting a server);
  only 15-20 calls are needed!

Great design

- Learn from real-world experience (e.g. EC2 added Elastic IPs, Elastic Block Store; we can build in)
- Agree simple semantics and simple syntax enabling cloud vendors and ecosystem to implement swiftly and developers to quickly learn and use the API

The case for great API design

Amazon EC2, ElasticHosts and GoGrid APIs offer similar functionality.
ElasticHosts and GoGrid demonstrate the power of a cleaner, simpler approach:

	Amazon EC2	ElasticHosts	GoGrid
Total calls in API	38	20	15
Starting a server with static IP and persistent drive	3 calls with ~1000 bytes of data	1 call with ~100 bytes of data	1 call with ~100 bytes of data
API documentation	300 pages	1 page overview	20 pages

Clean, simple APIs that developers
can quickly learn and use

Sources: http://awsdocs.s3.amazonaws.com/EC2/latest/ec2-dg.pdf, http://www.elastichosts.com/products/api, http://wiki.gogrid.com/wiki/index.php/Api

Design principles: Simple semantics

The API must be very fast for developers to learn and use. They should be able to get started with minimal documentation and a few examples.

Few powerful orthogonal commands

- Each call adds overhead, both in code and response times
- Produce a few powerful calls which do the work of many smaller ones – e.g. a single call for “start server”, rather than many to configure each aspect of the server

No artificial abstractions

- Hide internal implementation details wherever possible.
- Virtual server hardware should be specified in the well-known language of physical hardware – e.g. MHz of CPU cores, GB of RAM, GB of IDE hard drives

Immediate response where possible

- Almost all API commands should be synchronous, and should complete within seconds of all input data arriving

Summarized from: http://www.elastichosts.com/blog/2009/01/01/designing-a-great-http-api/

Design principles: Simple syntax

The API must be easy to call from a range of standard tools – e.g. from a single command at the Unix shell using the curl command line HTTP tool

Choice of syntax

- Commands should be available in XML, JSON and text "skins" for ease of use by all users

Use of internet standards

- Reuse standard HTTP mechanisms wherever possible: for security (SSL/TLS), authentication (basic auth), error codes (status codes), choice of "skins" (content-type/accept), etc.

Summarized from: http://www.elastichosts.com/blog/2009/01/01/designing-a-great-http-api/