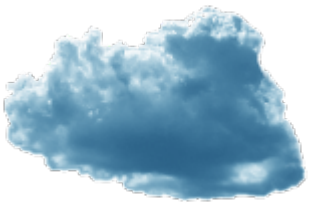
CloudI 

QUICKSTART | API | FAQ | Download | Source

A Cloud as an Interface

CloudI is an open-source private cloud computing framework for **efficient**, **scalable**, and **fault-tolerant** soft-realtime event processing. CloudI uses an Erlang core, with its actor model, to be a universal integrator for services, databases, and messaging buses. All CloudI services exchange unstructured data within a Service Oriented Architecture (SOA) with RESTful development practices. CloudI service requests are publish/subscribe or point-to-point with simple Access Control Lists (ACLs), highly configurable load balancing, and service based fault-tolerance. CloudI scales linearly for distributing realtime service load and can easily integrate with software written in C/C++, Erlang, Java, Python, and Ruby with its light-weight API. Read more about what CloudI is within the FAQ.

CloudI Quick Start

Install Quick Start Tools

- **Ubuntu:** sudo apt-get install wget curl
- **OSX:** sudo port install wget curl

Get CloudI Running

```
$ wget http://sourceforge.net/projects/cloudi/files/latest/download -O cloudi-1.2.0.tar.gz
$ tar zxvf cloudi-1.2.0.tar.gz
$ cd cloudi-1.2.0/src
$ ./configure
$ make
$ sudo make install
$ cd ../..
$ sudo cloudi start
```

The CloudI tests are now running and consuming your available CPUs.

The Quick Start guide below shows how to create both an internal (Erlang) CloudI service and an external (Python) CloudI service.

Create an Internal (Erlang) CloudI Service

```
$ mkdir cloudi-quickstart
$ cd cloudi-quickstart
$ cat << EOF > hello_world.erl
-module(hello_world).
-behaviour(cloudi_service).

%% cloudi_service callbacks
-export([cloudi_service_init/3,
         cloudi_service_handle_request/11,
         cloudi_service_handle_info/3,
         cloudi_service_terminate/2]).

-include_lib("cloudi_core/include/cloudi_logger.hrl").

-record(state,
    {
    }).

cloudi_service_init(_Args, _Prefix, Dispatcher) ->
    cloudi_service:subscribe(Dispatcher, "hello_world/get"),
    {ok, #state{}}.

cloudi_service_handle_request(_Type, _Name, _Pattern, _RequestInfo, _Request,
                              _Timeout, _Priority, _TransId, _Pid,
                              #state{} = State, _Dispatcher) ->
    {reply, <<"Hello World!">>, State}.

cloudi_service_handle_info(Request, State, _) ->
    ?LOG_WARN("Unknown info \"~p\"", [Request]),
    {noreply, State}.

cloudi_service_terminate(_, #state{}) ->
    ok.
EOF
$ erlc -pz /usr/local/lib/cloudi-1.2.0/lib/cloudi_core-1.2.0 hello_world.erl
./hello_world.erl:2: Warning: behaviour cloudi_service undefined
```

You now have a compiled internal CloudI service which is ready to run. You can also provide an OTP application file with the same name, if the internal CloudI service has application dependencies.

Run the Internal (Erlang) CloudI Service

While you are still in the cloudi-quickstart directory, use the CloudI Service API to run the internal CloudI service.

```
$ curl -X POST -d '"'`pwd`'"' http://localhost:6467/cloudi/api/erlang/code_path_add
$ cat << EOF > hello_world.conf
[{internal,
  "/quickstart/hello/",
  hello_world,
  [],
  lazy_closest,
  5000, 5000, 5000, [api], undefined, 1, 5, 300, []}]
EOF
$ curl -X POST -d @hello_world.conf http://localhost:6467/cloudi/api/erlang/services_add
```

These HTTP requests communicate with src/lib/cloudi_services_internal /src/cloudi_service_http_cowboy.erl which runs the cowboy HTTP webserver on port 6467, because of the default CloudI configuration (installed at /usr/local /etc/cloudi/cloudi.conf). The request becomes a CloudI request, within the cloudi_service_http_cowboy internal CloudI service, which is sent to src/lib /cloudi_services_internal/src/cloudi_service_api.erl. The cloudi_service_api internal CloudI service provides runtime configuration of CloudI.

You will notice that the syntax used to start the CloudI service in the hello_world.conf file is the same as what is specified in the "services" section of /usr/local/etc/cloudi/cloudi.conf.

Use the Internal (Erlang) CloudI Service

```
$ curl http://localhost:6467/quickstart/hello/hello_world
Hello World!
```

The HTTP GET request has received the "Hello World!" message from your new internal CloudI service.

You can get the same behavior with an external CloudI service, which is written in a supported programming language, currently: C/C++, Java, Python, or Ruby.

Use an External (Python) CloudI Service

```
$ cat << EOF > hello_world.py
import sys
sys.path.append('/usr/local/lib/cloudi-1.2.0/api/python/')
from cloudi_c import API

class Task(object):
    def __init__(self):
        self.__api = API(0) # first/only thread == 0

    def run(self):
        self.__api.subscribe("hello_world_python/get", self.__hello_world)
        result = self.__api.poll()
        print 'exited:', result

    def __hello_world(self, command, name, pattern, request_info, request,
                      timeout, priority, trans_id, pid):
        return 'Hello World!'

if __name__ == '__main__':
    assert API.thread_count() == 1 # simple example, without threads
    task = Task()
    task.run()
EOF
$ PYTHON_PATH=`which python`
$ PWD=`pwd`
$ cat << EOF > hello_world_python.conf
[{external,
  "/quickstart/hello/",
  "$PYTHON_PATH",
```

```
  "$PWD/hello_world.py",
  [],
  none, tcp, 16384,
  5000, 5000, 5000, [api], undefined, 1, 1, 5, 300, []}]
EOF
$ curl -X POST -d @hello_world_python.conf http://localhost:6467/cloudi/api/erlang/services_add
$ curl http://localhost:6466/quickstart/hello/hello_world_python
Hello World!
```

You may notice the port number 6466 is different from what was used for the internal CloudI service. This is a different instance of the cloudi_service_http_cowboy internal CloudI service which forces all outgoing CloudI requests to be binary. All external CloudI services handle request data and request_info data as binary data, to simplify integration efforts and make service runtime more efficient. If you had tried to use the port number 6466 for the CloudI Services API, you would have received a timeout, not because binary requests are not accepted, but rather because the cloudi_service_http_cowboy ACL (Access Control List) prevents API requests (with a service name pattern, referred to as api). Please refer to the API documentation for more information.

You now have an external CloudI service written in Python which is able to perform the same task as your internal CloudI service (written in Erlang). You can use the same techniques to create other external CloudI services with new or pre-existing source code to gain fault-tolerance and scalability. Creating CloudI services makes integration tasks simpler and allows your software to grow without limitations!