CloudI Integration Framework

Chicago Erlang User Group – May 27, 2015

Speaker Bio

- Bruce Kissinger is an Architect with Impact Software LLC.

IMPACT SOFTWARE

- Linkedin: https://www.linkedin.com/pub/bruce-kissinger/1/6b1/38
- Email: brucekissinger at gmail dot com

Agenda

- What is CloudI?
- How Do You Use It?
- Should You Use CloudI On Your Next Project?

What Is CloudI?

CloudI Definition

- CloudI is an open-source integration cloud that can be deployed publicly or privately. It supports the development of services that can be created in many different programming languages and provides scalability and fault-tolerance.

Cloud Computing

- Essential Characteristics
  - On Demand Self Service – provision computing resources without requiring human intervention from the service provider
  - Broad Network Access – capabilities are available over the network and accessed using standard mechanisms
  - Resource Pooling – can service multiple consumers using a multi-tenant model with different resources dynamically assigned based on demand
  - Rapid Elasticity – rapid provisioning and scaling of resources
  - Measured Service – resource usage can be monitored, controlled, and reported

(Source: NIST Cloud Computing Definition, 2012)

CloudI Alignment

Cloud Characteristic	CloudI	Comments
On-Demand Self Service	✓	Resources controlled via HTTP request
Broad Network Access	✓	Uses standard network protocols
Resource Pooling	✓	Provided by underlying Erlang/OTP capabilities
Rapid Elasticity	✓	Provided by underlying Erlang/OTP capabilities
Measured Service	Partial	Timeouts, queue depth, and other parameters measured. Limited built-in reporting capabilities

Service Oriented Architecture

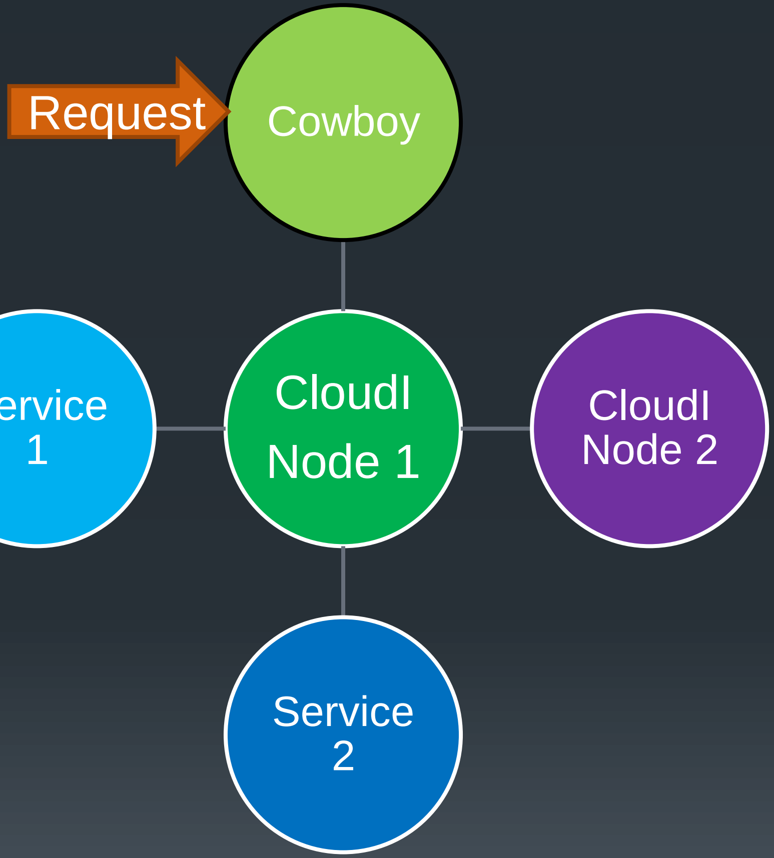
- Definition – a set of principles and methodologies for designing and developing software in the form of interoperable services. (Source: Wikipedia)
- Service - discrete unit of business functionality that is made available through a service contract. This contract specifies all interactions between the service consumer and service provider.
- Common Service Characteristics
  - Encapsulated – hide the service implementation details
  - Different Levels of Granularity – coarse-grained services provide greater level of functionality within a single service operation. Fine-grained services perform a single specific task.
  - Stateless – do not remember the last thing they did nor care what the next is
  - Location and Language Independent – accessible to any authorized user on any platform, from any location
  - Modular – services are self contained and autonomous

CloudI Alignment

Service Characteristic	CloudI	Comments
Encapsulated	✓	Service contract defined using configuration property list
Different Levels of Granularity	✓	Coarse and fine grained services supported equally
Stateless	✓	Use of a RESTful API protocol helps enforce statelessness
Location and Language Independent	✓	Services can run on specific or all cluster nodes. Supports 10 programming languages
Modular	✓	Services are run in OS processes with an Erlang thread monitoring them

CloudI Architecture

- A separate operating system process is used to isolate each non-Erlang service
- A separate Erlang process is associated with each OS process for monitoring and control
- CloudI message bus provides security and location transparency
- CloudI leverages Erlang/OTP internally



CloudI Language Bindings

- Erlang
- Elixir
- C / C++
- Java
- JavaScript / Node.js
- Perl
- PHP
- Python
- Ruby

Built-In Services

- Filesystem – provides file read, write, notification functions
- HTTP Client – handles HTTP REST requests
- HTTP Servers – Cowboy and Elli
- OAuth – open authorization standard
- TCP – socket communication using TCP protocol
- UDP – socket communication using UDP protocol
- Timers – send messages with timer behavior
- Quorum – used to provide fault tolerance across distributed services
- Queue – persistent queue that survives restarts
- ZeroMQ – high-performance message library
- Elasticsearch – distributed full-text search server
- Map/Reduce service – fault tolerant, database agnostic

Built-In Database Services

- Database integration services
  - MySQL
  - PostgresSQL
  - Memcached
  - Riak
  - Couchdb
  - Cassandra DB and CQL
  - Tokyo Tyrant
  - Generic in-memory

CloudI API – Controlling the Cloud

- Access Control Lists
  - Add or remove an ACL entry
  - List ACL entries
- Service
  - Add, Remove, or Restart a service
  - List the subscriptions for a service instance
  - List service configuration for a given service name
  - List all services
- Nodes
  - Set Configuration – can use Erlang or Amazon Web Services (AWS) node discovery
  - Add or remove a node
  - List all nodes, alive nodes, or dead nodes
- Logging
  - Set logging file
  - Set logging level
  - Set logging format
  - Set log redirection
  - List configuration
- Code Path
  - Add or remove a code path entry
  - List code paths

CloudI API – Service Control

- Initialization / Termination – starts service and provides orderly shutdown
- Subscribe – subscribe to a service name pattern
- Unsubscribe – remove the subscription for a service name
- Send Sync – send a synchronous request to a service
- Send Async – send an asynchronous request to a service and get a transaction id
- Forward - forward the service request to a different destination, possibly with different parameters
- Mcast Async - send the service request asynchronously to all services that have subscribed to a name pattern and gets a list of transaction ids
- Return - return a response to a service request
- Receive Async - receive an asynchronous service request's response
- Poll - accept service requests while blocking execution until either the timeout value expires or the service terminates

How Do You Use CloudI?

Simple as 1, 2, 3

1. Add message subscriptions and handler templates to existing code and compile
2. Create a configuration file
3. Register the service

Erlang – Export Functions

```
-module(book).
-behaviour(cloudi_service).

%% cloudi_service callbacks
-export([cloudi_service_init/4,
        cloudi_service_handle_request/11,
        cloudi_service_handle_info/3,
        cloudi_service_terminate/3]).
```

Erlang – Service Initialization

```
cloudi_service_init(_Args, _Prefix, _Timeout, Dispatcher) ->

  % subscribe to different request patterns
  cloudi_service:subscribe(Dispatcher, "newbooks/get"),
  cloudi_service:subscribe(Dispatcher, "popularbooks/get"),

  % return ok
  {ok, #state{}}.
```

Erlang – Handling Requests

```
cloudi_service_handle_request(Type, Name, Pattern, _RequestInfo, Request,
                    _Timeout, _Priority, _TransId, _Pid, #state{} = State, Dispatcher) ->

    % based on the pattern and request, perform the appropriate action

    case Pattern of
        "/recommend/book/newbooks/get" ->
            ReplyRecord = find_new(Dispatcher);        % find_new is a local function

        "/recommend/book/popularbooks/get" ->
            ReplyRecord = find_popular(Dispatcher);    % find_popular is a local function

        _ ->
            ReplyRecord = cloudi_x_jsx:encode(["Invalid Request"])
    end,

    % send reply
    {reply, ReplyRecord, State}.
```

Erlang – Calling Another Service

```
…
Query = "select id, title from items",

Status = cloudi_service:send_sync(Dispatcher,
  "/db/mysql/book",
  <<>>,
  Query,
  undefined,
  undefined),

case Status of
  {ok , Result} ->
    Json_result = parse_items(Result);
  _ ->
    Json_result = cloudi_x_jsx:encode(<<"No data found">>)
  end,

Json_result.
```

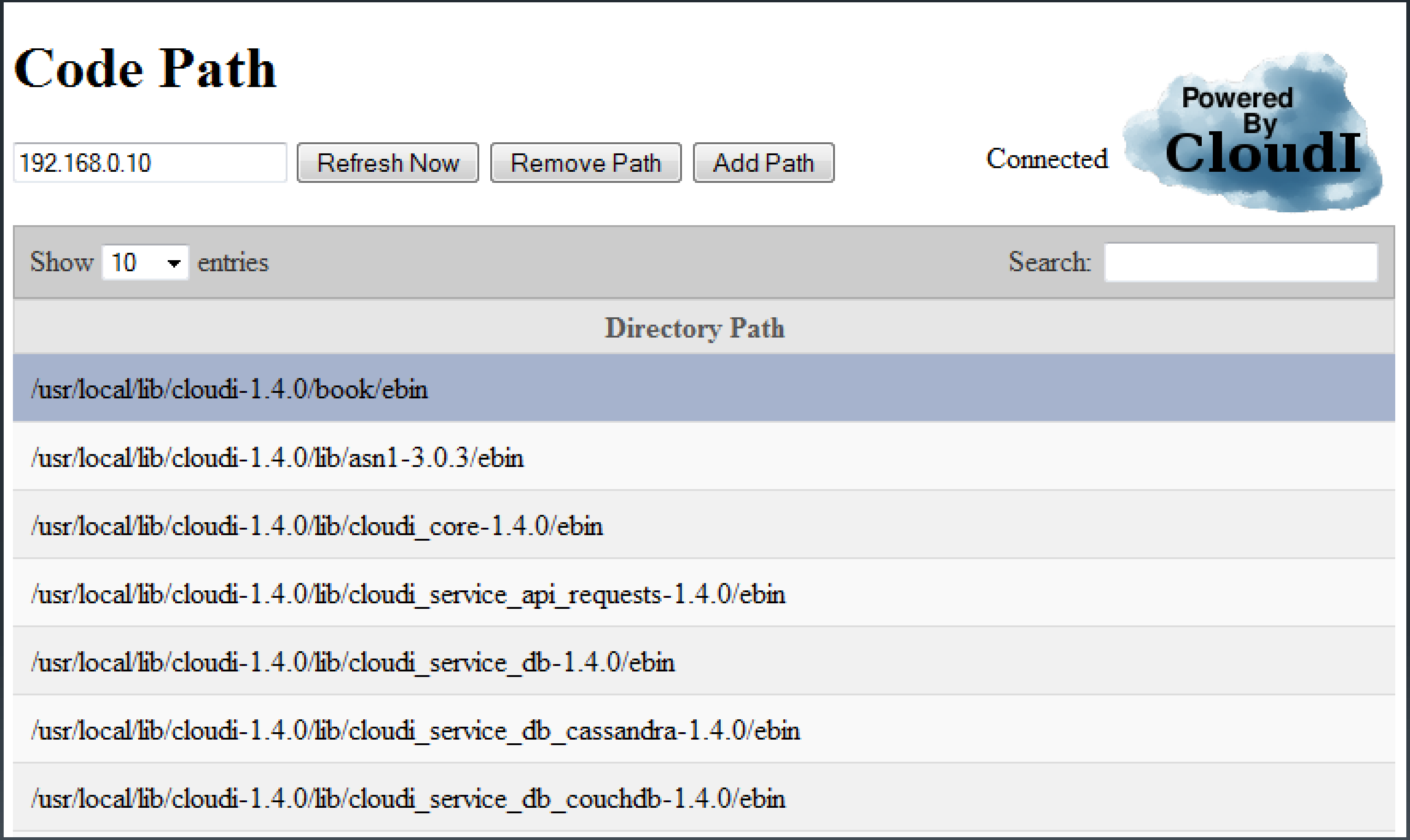
Erlang – Service Configuration

```
[{internal,
  "/recommend/book/",                % Service name
  book,                              % Erlang module
  [],
  immediate_closest,
  5000, 5000, 5000, undefined, undefined, 1, 5, 300,
  [{reload, true}, {queue_limit, 100}]
}]
```

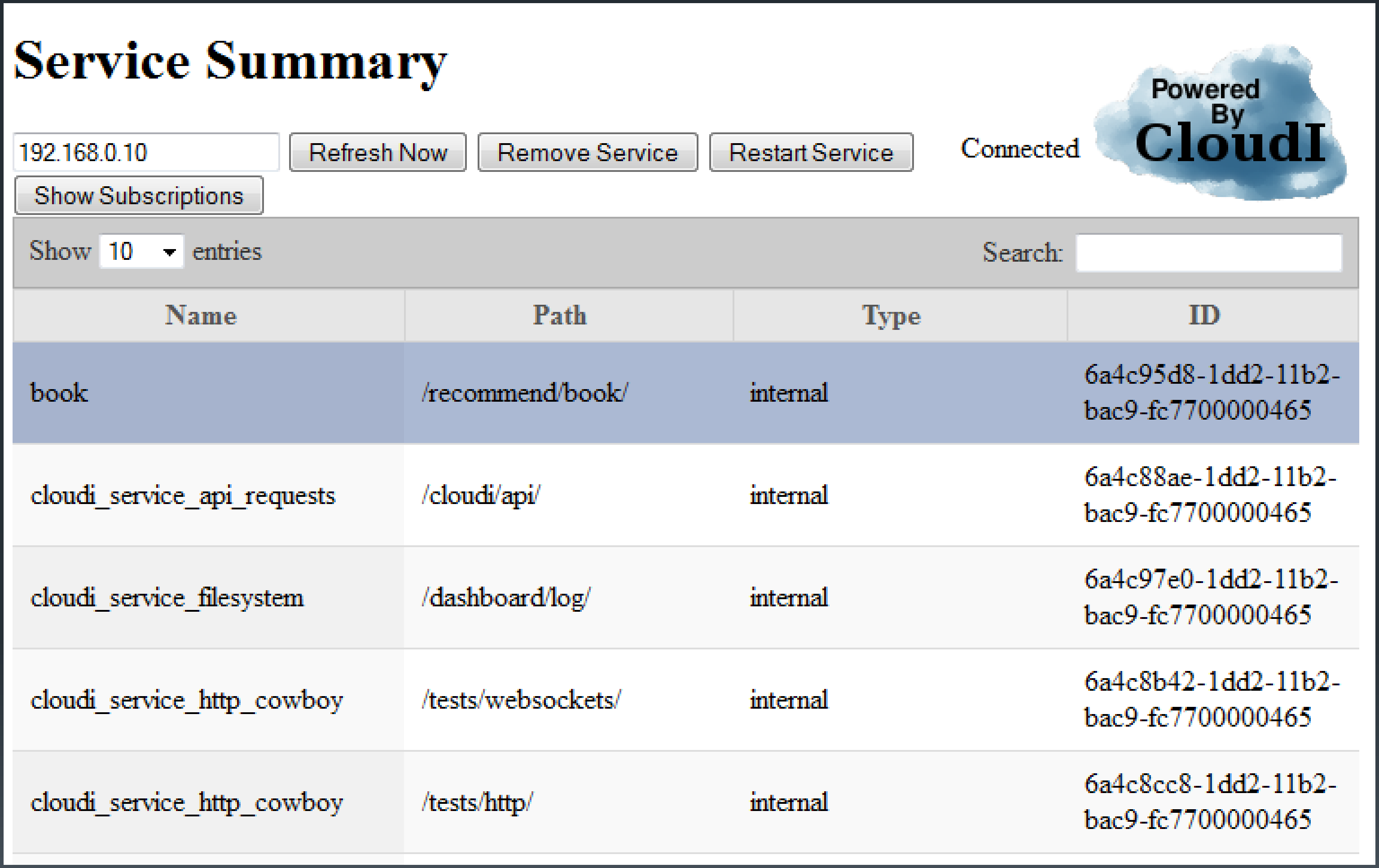
Erlang – Registering the Service

```
CLOUDI_HTTP=http://localhost:6467/cloudi/api/erlang

# Add the directory where the complied code is located
curl -X POST -d @path.conf
        $(CLOUDI_HTTP)/code_path_add

# Add the service
curl -X POST -d @book.conf
        $(CLOUDI_HTTP)/services_add
```

Dashboard Examples

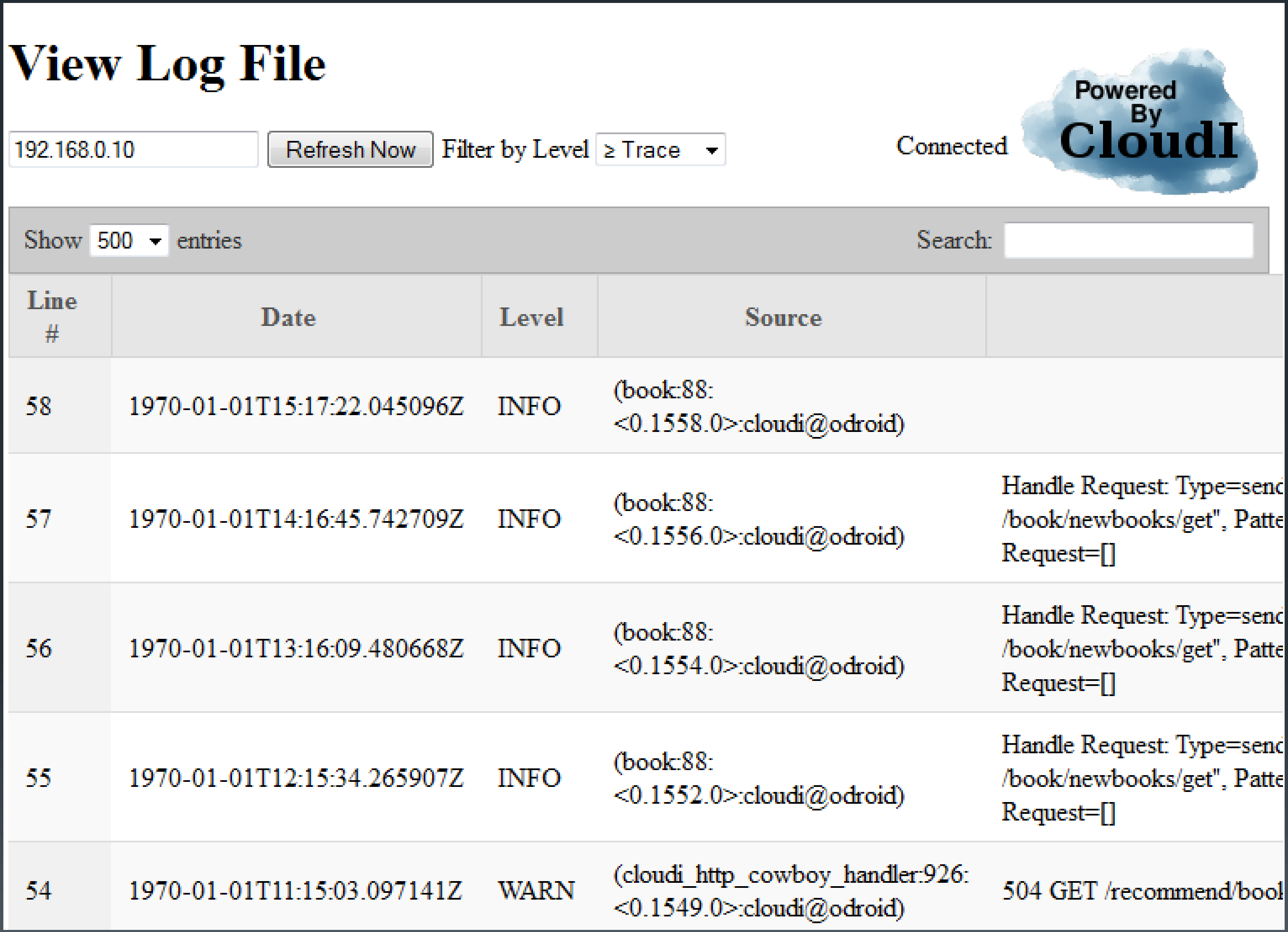


Directory Path
/usr/local/lib/cloudi-1.4.0/book/ebin
/usr/local/lib/cloudi-1.4.0/lib/asn1-3.0.3/ebin
/usr/local/lib/cloudi-1.4.0/lib/cloudi_core-1.4.0/ebin
/usr/local/lib/cloudi-1.4.0/lib/cloudi_service_api_requests-1.4.0/ebin
/usr/local/lib/cloudi-1.4.0/lib/cloudi_service_db-1.4.0/ebin
/usr/local/lib/cloudi-1.4.0/lib/cloudi_service_db_cassandra-1.4.0/ebin
/usr/local/lib/cloudi-1.4.0/lib/cloudi_service_db_couchdb-1.4.0/ebin



Service Summary

Name	Path	Type	ID
book	/recommend/book/	internal	6a4c95d8-1dd2-11b2-bac9-fc7700000465
cloudi_service_api_requests	/cloudi/api/	internal	6a4c88ae-1dd2-11b2-bac9-fc7700000465
cloudi_service_filesystem	/dashboard/log/	internal	6a4c97e0-1dd2-11b2-bac9-fc7700000465
cloudi_service_http_cowboy	/tests/websockets/	internal	6a4c8b42-1dd2-11b2-bac9-fc7700000465
cloudi_service_http_cowboy	/tests/http/	internal	6a4c8cc8-1dd2-11b2-bac9-fc7700000465



Filter by Level ≥ Trace

Connected

Line #	Date	Level	Source	
58	1970-01-01T15:17:22.045096Z	INFO	(book:88: <0.1558.0>:cloudi@odroid)	
57	1970-01-01T14:16:45.742709Z	INFO	(book:88: <0.1556.0>:cloudi@odroid)	Handle Request: Type=senc /book/newbooks/get", Patte Request=[]
56	1970-01-01T13:16:09.480668Z	INFO	(book:88: <0.1554.0>:cloudi@odroid)	Handle Request: Type=senc /book/newbooks/get", Patte Request=[]
55	1970-01-01T12:15:34.265907Z	INFO	(book:88: <0.1552.0>:cloudi@odroid)	Handle Request: Type=senc /book/newbooks/get", Patte Request=[]
54	1970-01-01T11:15:03.097141Z	WARN	(cloudi_http_cowboy_handler:926: <0.1549.0>:cloudi@odroid)	504 GET /recommend/bool

Java Service Example

- The general steps for adding a Java application to CloudI are:
  - Create a new class named *Main* that will initialize the CloudI API
  - Create a new class named *Task* that subscribes to various CloudI requests and delegates the processing of these requests to different Java methods
  - Create a JAR file that contains the different Java classes
  - Add the JAR file to the CloudI configuration

Java – Main Class

```
import org.cloudi.API;

public class Main {
  public static void main(String[] args) {
      try {
          final int thread_count = API.thread_count();
          assert (thread_count == 1);
          Task t = new Task(0);
          t.run();
      } catch (API.InvalidInputException e) {
          e.printStackTrace(API.err);
      }
  }
}
```

Java – Task Class – Part 1

```
import com.ericsson.otp.erlang.OtpErlangPid;
import java.io.UnsupportedEncodingException;
import org.cloudi.API;

public class Task {
  private API api;

  public Task(final int thread_index) {
      try {
          this.api = new API(thread_index);
      } catch (API.InvalidInputException e) {
          e.printStackTrace(API.err);
          System.exit(1);
      } catch (API.MessageDecodingException e) {j
          e.printStackTrace(API.err);
          System.exit(1);
      } catch (API.TerminateException e) {
          System.exit(1);
      }
  }
```

Java – Task Class – Part 2

```
public void run() {

    try {

        // subscribe to different CloudI services
        this.api.subscribe("load_catalog/get", this, "startLoadCatalog");
        this.api.subscribe("generate_ratings/get", this, "startGenerateRatings");
        this.api.subscribe("load_predictions/get", this, "startLoadPredictions");

        // accept service requests
        this.api.poll();

    } catch (API.TerminateException e) {
        API.err.println("Book Utilities TerminateException caught " + e.getMessage());
    } catch (Exception e) {
        API.err.println("Book Utilities Exception caught " + e.getMessage());
    }
}
```

Java – Calling Another Service

```
…
byte[] service_request =
        ("SELECT max(quantity) FROM items").getBytes();

org.cloudi.API.Response response =
        api.send_sync("/db/mysql/book", service_request);
…
```

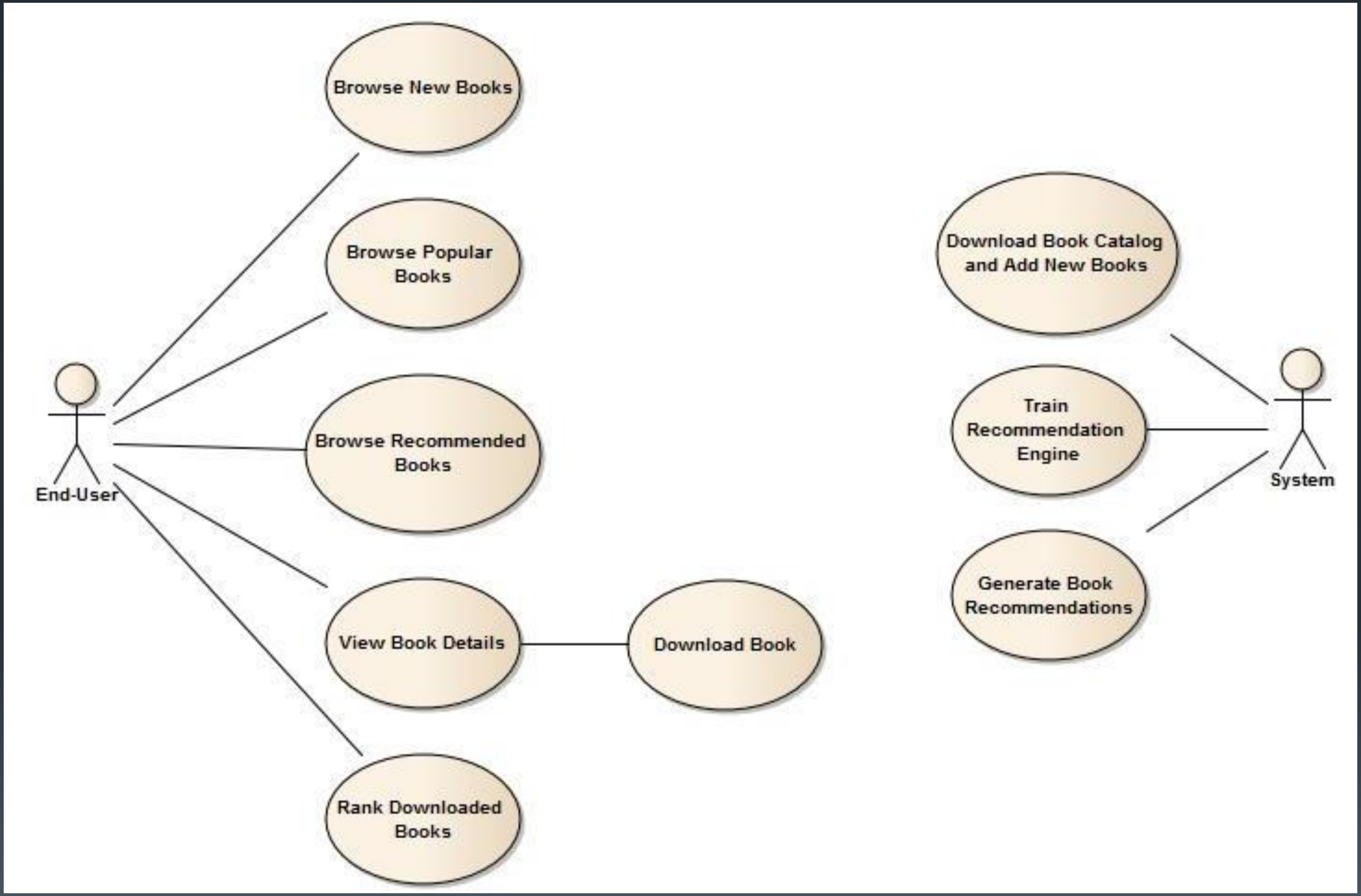
Java – Service Configuration

```
[
 {external,
                "/book/utility/",                          % service name
                "/opt/java/jdk1.7.0_05/bin/java",
                "-cp /usr/local/lib/cloudi-1.5.0/api/java/ "
                "-ea:org.cloudi... -jar
/home/bruce/Projects/BookUtilities/deploy/BookUtilities.jar",
                [],
                lazy_closest, tcp, default,
                50000, 50000, 50000, undefined, undefined, 1, 1, 5,
300, []
        }
]
```

Simple as 1, 2, 3, 4, 5, 6, 7

1. Design the message API
2. Design the message data structures – especially if using mixed languages
3. Add message subscriptions and handler templates to existing code and compile
4. Create a configuration file
5. Register the service
6. Repeat Step 5 for all nodes in the cluster
7. Measure performance and fine tune the service configuration

Design the Message API – Part 1



Design the Message API – Part 2

Use Case	Method	URL
Browse New Books	GET	/book/newbooks
Browse Popular Books	GET	/book/popularbooks
Browse Recommended Books	GET	/book/recommendedbooks?user=X
View Book Details	GET	/book/allbooks?id=X
Download Book	GET	/book/download?id=X&user=Y
Create New User	GET	/book/newuser
Get Unrated Books	GET	/book/unrated?user=X
Rank Downloaded Book	POST	/book/download/
Add Book to Collection	POST	/book/allbooks/

Should You Use CloudI On Your Next Project?

Strongly Consider

- If your project needs cloud-type characteristics
  - On Demand Self Service
  - Broad Network Access
  - Resource Pooling
  - Rapid Elasticity
- Project deployed to a internal or external cloud
  - CloudI has strong support for Amazon cloud
- If your project uses a service-oriented architecture style
  - Set of principles and methodologies for designing and developing software in the form of interoperable services
- If you can leverage the built-in services
- If you are using a mix of languages
- If you need Erlang-style fault tolerance with these languages

Investigate More

- If you are develop completely in Erlang/OTP, CloudI can still offer some benefits including:
  - Use of CloudI built-in services
  - A service container abstraction for simpler Service Oriented Architecture development.
  - Finer control of service start order and runtime characteristics
  - See http://www.cloudi.org/faq.html#4_Erlang for list of other potential benefits

Probably Not For You

- If you do not use a service-oriented architecture style
- If you need very robust service or message security
  - CloudI does not implement role-based security for calling services
  - CloudI does not use secure encrypted messages
- If you need very large scale clusters
  - CloudI relies on Erlang/OTP for cluster management & communication
  - Practical limit is < 100 nodes
- If your project is deployed on Windows-based operating systems
    - In theory this is possible, but installation might be challenging

Additional References

- Project site – http://cloudi.org
- Mailing list - http://groups.google.com/group/cloudi-questions
- CloudI Tutorial - http://www.impactsoftwarelabs.com/cloudi

Questions?