A Cloud as an Interface
to All Your Source Code!
ErLounge, Vancouver BC
June 2\textsuperscript{nd} 2011

Michael Truog
mjtruog@gmail.com
What Is CloudI?

• A flexible application server that supports:
  • Many programming languages
  • Many messaging buses
  • Many databases
  • Scalable process management
  • Fault-tolerance with process isolation

• A Private Cloud Computing Solution

• A Scalable Server for Public Deployment of Online Services
Why Use CloudI?

- To make source code more scalable
- To manage unstable source code that offers critical functionality
  - Provide a method to migrate away from a dependency on unstable source code
- To reuse source code from diverse programming languages
- To simplify development with a flexible integration framework
What Is CloudI For?

- Event Handling
  - Websites, Games, ...
- Data Processing
  - Text processing, numerical computations, ...
- System Integration
  - Data routing, access control, ...
How To Develop A CloudI Service

- The CloudI API with: Ruby, Python, C/C++, Java or Erlang
  - Service Messaging: send_sync, send_async, recv_async, mcast_async
  - Service Advertising: subscribe, unsubscribe
  - Service Message Result: return, forward
- Passively accepts incoming service messages to resources (names) advertised by the service
How To Develop A CloudI Service...

- A service name prefix is provided by the service configuration, along with thread count and process count.
- All subscribe API calls specify a service name suffix, so many destinations within a single service share a common service name prefix.
- All service messaging utilizes the complete service name as a destination.
- Service messages are automatically load-balanced based on the sending service's config.
Configuration

• Dynamic configuration is supported by the Cloud1 Job API
  • Accessible by JSON-RPC, HTTP, and services
• Access Control Lists (ACL) specify service name destination prefixes that are explicitly allowed or denied (affects sending)
• Service command line, restart characteristics, threads, processes, ACLs, load-balancing, etc., is all defined as a Job to execute
Integration Considerations

- Databases, HTTP, ZeroMQ, etc., integration occurs with separate Erlang CloudI services
- No data format or type is enforced for incoming service messages
  - Non-Erlang (i.e., external) service messages are received as binary data
- ACL definitions can easily isolate a service
- Global data can be stored within a database
- External source code uses service messaging for input and output (stdout and stderr get logged)
Migration To Cloud

1) Develop or modify source code to create a service that handles input as service messages and returns output

2) Add service configuration to create the Job based on load-balancing, fault-tolerance, isolation (ACL), and capacity requirements

3) Any source code that is a scalability bottleneck can be gradually migrated to Erlang as a separate service using the same service name
More Information

- Frequently Asked Questions
  http://cloudi.org/faq.html

- Mailing List
  http://groups.google.com/group/cloudi-questions

- CloudI Expert
  mjtruog@gmail.com
Questions?