FPComplete

Bringing Haskell to the World

www.fpcomplete.com

Experience Report

Building Haskell Development and Deployment tools

Gregg Lebovitz

Director, Product Management

FP Complete

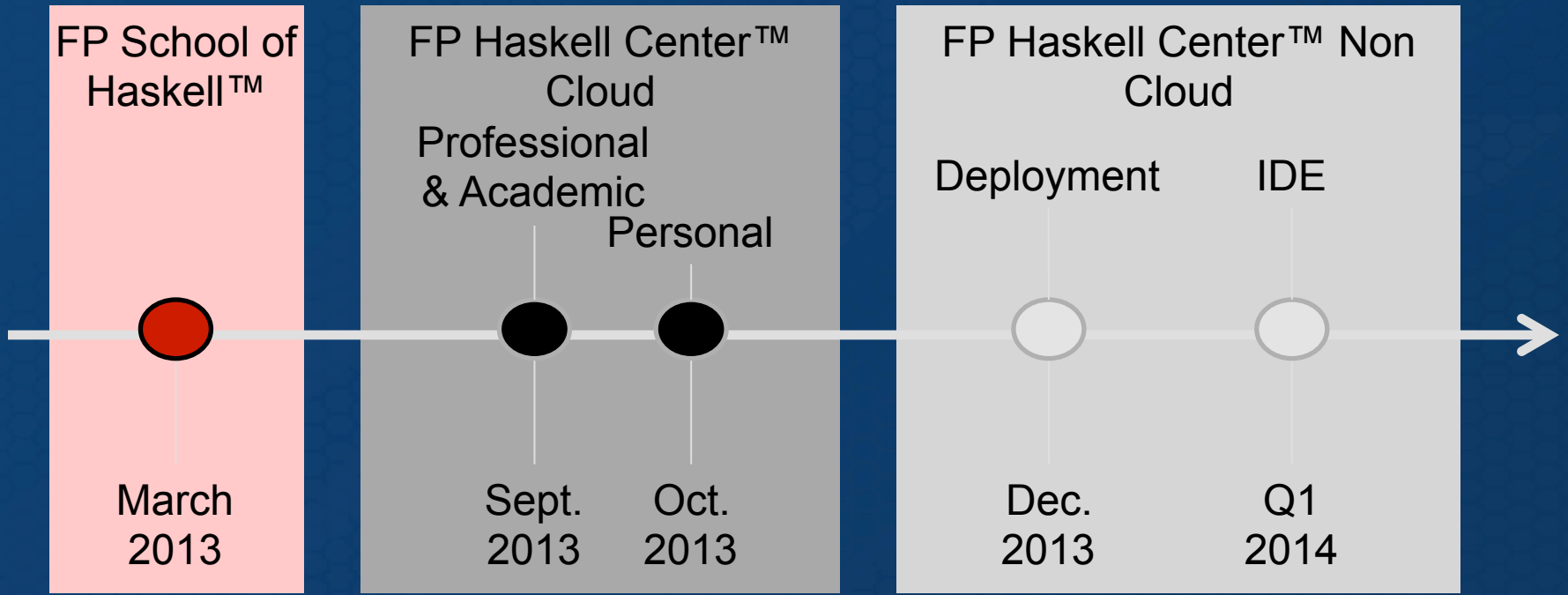
Company Goals

- Increase Haskell Adoption
- Make Haskell More Accessible
- Offer Commercial Grade Tools and Support
- Simplify Haskell Deployment
- Support the Haskell Community
- Leverage what the Community Offers

FP Haskell Center

- Web Front End
  - No Struggle Setup
  - Access to Help
  - Easy to Integrate with Haskell
- Haskell Back End
  - Project Management
  - Provide Developer Feedback
  - Build System
  - Cloud Base Execution and Deployment
- Cloud allows faster product evoluton

Product Roadmap



UI

- UI Details
  - Backend Implemented in Yesod
  - Lots of Library Support (conduits, etc)
  - UI Uses Javascript (using Fay)
  - Heavy lifting done in Backend
- Very few Issues surfaced

Product Goals

- Web Access (initially)
- Live Feedback To Developer
- Point and Click Build Process
- Simple Project Management
- Access to Source Repositories
- Integrated Help and Documentation

Challenges

- Javascript Coding Issues
- Stable set of libraries
- Compiler Integration (feedback and errors)
- Integrating with git
- Running in the cloud
- Deploying Applications
- Billing system integration

Stability Issues

- Fay - Javascript
  - Eliminated most Javascript issues
  - Allowed us to focus on features not bugs
- Create Stackage
  - Managed by Authors
  - Packages must be version compatible
  - Libraries are vetted, and tested
  - Commercial Support for Customers

Code Analysis

- Integrating GHC Via Library
  - Access to the Abstract SyntaxTree
  - Report Errors
  - Map Source Location to AST
  - Locate where Identifiers are Defined
  - Get Details about Types and Identifiers
  - Support Auto-complete
- Do the same for HLint

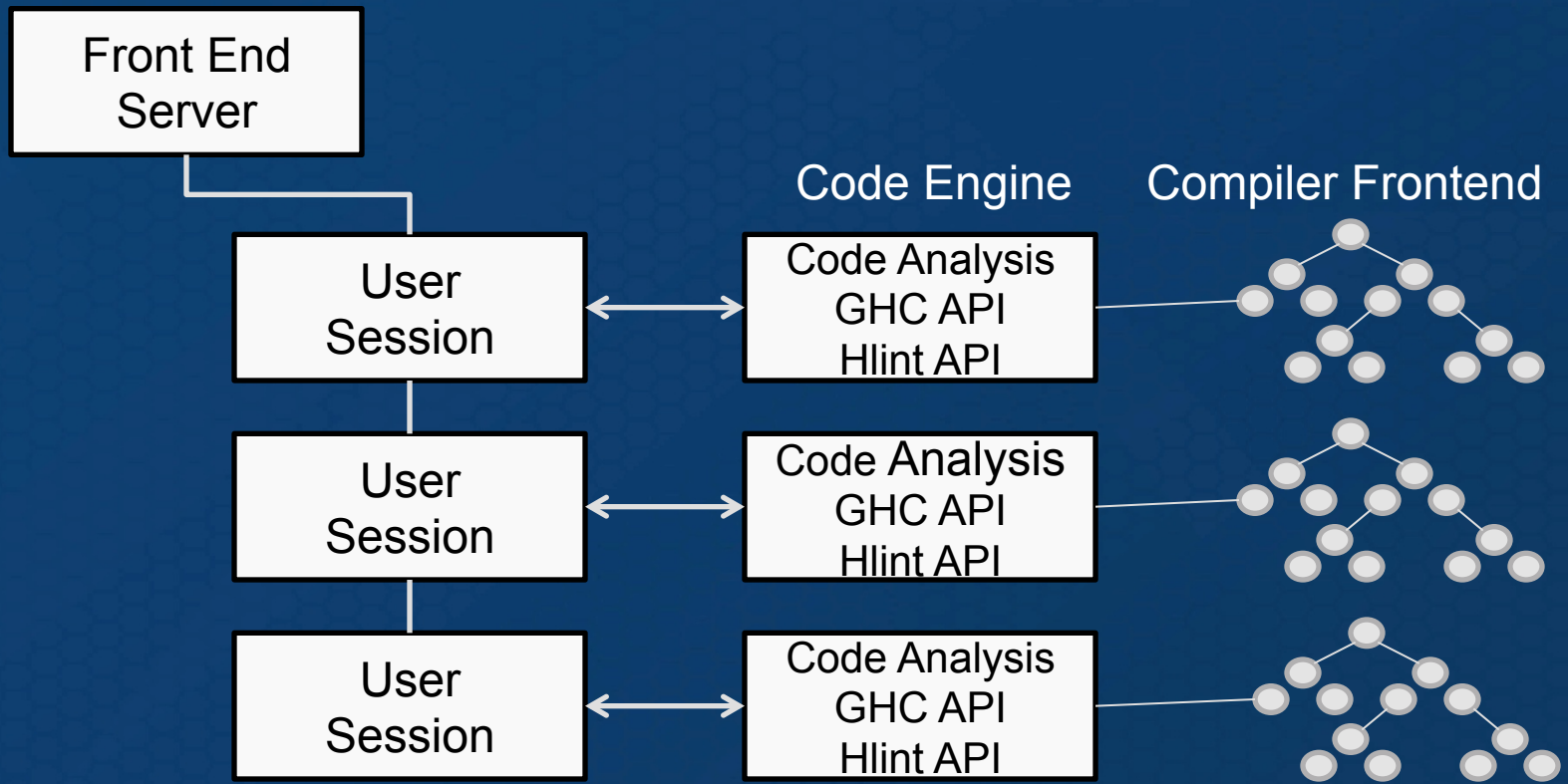
Beyond Errors (Future)

- Once You have the Compiler Front End
  - Do Syntax Analysis
  - Recommend Code Improvements
  - Track Code Execution
  - Implement Debugging
  - Add Profiling Information
  - Improve Error Reporting
  - Understand performance issues

Responsiveness and Stability

- Challenges
  - Do code analysis
  - Provide Lots of Live User Feedback
  - But Make the UI “Snappy”
- Solution
  - Separate the Web Front End and
  - Code Analysis Engines

Separate Processes

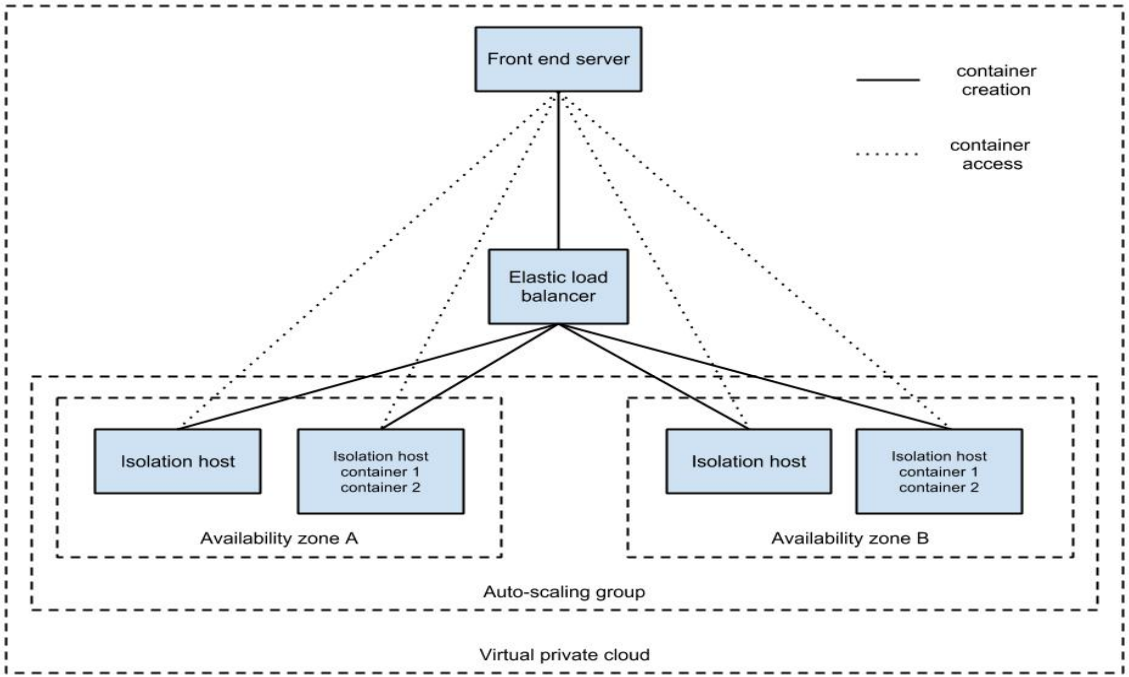


Running in the Cloud

- Use LXC to create Isolation Containers
  - Each container is a mini machine
  - Includes a full runtime environment
  - Runs required system services
  - Provides ephemeral storage
  - Containers can run on shared systems
  - Share underlying resources to reduce footprint
    - OS, Libraries, and System Services

Containers Distributed as

- Containers on dedicated and shared systems



IDE Uses Isolation Containers

- Front End handles requests from IDE
- Initiates a User sandbox container
- Loads Environment from Persistent Storage
  - Includes Active files
  - Project Settings
  - Previous State Settings
- User work is Saved to Persistent Storage (S3)

Managing Projects

- Visual Representation of Projects
- Projects Stored in Git Repositories
- Contains Project Settings/Definition File
- Repo Access Through Haskell gitlib-2
  - Haskell robustness
  - Multiple backends
    - Git C library backend
    - GitHub C library backend
    - IDE Local repository stored in S3
    - Others?

Building Projects

- IDE Code Generated by Backend Process
  - Uses Active GHC Front End
  - Generates Bytecode
  - Runs Bytecode in GHC Frontend Container
  - Exceptions leave IDE intact
- Deployment Build System Uses Cabal APIs
  - Import existing cabal files
  - Preprocess CPP Macros
  - Build executables for deployment
  - Generate licenses for deployment executables

Deploying Projects

- Haskell Has No Standard Way To Deploy Apps
- We Constructed A Deployment System
  - Compile Source to Executables
    - Haskell Libraries Linked Statically
  - Create Isolation Container
  - Install FP Application Server
  - Launch Instance (dedicated or shared)
  - Load Executable
  - Start Configuration Manager
  - Use Keter and Chef to Keep Things Running

Billing

- Billing Processor Provides SOAP APIs
  - Haskell SOAP Library Not Complete
  - Processor Supports gsoap
- Gsoap generates C++ from WSDL Files
  - FFI Requires C Bindings
  - Must generate Isomorphic mappings to C++ data
  - Fortunately all Gsoap data delivered as strings
- Limitations in GHC, Cabal, Linux made hard

Summary

- Haskell made development easier
  - Fewer Errors
  - Robust Code
- Our tools reduced our development effort
  - Stackage for Compatible Libraries
  - Integrated Code Analysis Tools
  - Containers used everywhere for running code
  - Code, Build and Deploy
- Haskell requires more commercial libraries
  - Billing Engine That Only Talks SOAP