School of Haskell

Haskell Development Center

Gregg Lebovitz

Director, Client Projects

FP Complete

FPComplete

Agenda

- Explain Design Goals
- Tell the story behind School of Haskell
- Demonstrate current implementation
- Give overview of School architecture
- Discuss the service architecture
- Present our Next Steps
- Provide some target milestones

Background

- FP Complete launched March 2012
- Goal: Create a commercial Haskell platform
- Current plan:
  - SaaS develop and deploy in the cloud
  - Option to deploy on private cloud or VM
  - Near future: develop on private cloud or VM
  - Further out: configuration, monitoring, exception tracing, interoperability, distributed applications.

FPComplete

FP Complete Team

- Aaron Contorer – Founder and CEO
- Bartosz Milewsky – Chief Architect
- Michael Snoyman – Engineering Lead
- Gregg Lebovitz – Product Management, Sales

Haskell Development Center

Four Ingredients

- Learn
- Develop
- Build
- Deploy

FPComplete

Original Ideas

- Free Education IDE for Interactive Learning
- Commercial IDE
  - Cloud Deployment
  - Deployment on a Private VM or Private Cloud
  - Private VM version of the IDE
  - Desktop version of the IDE
- Enhanced Deployment Options
  - Application Services

FPComplete

Current Roadmap

- School of Haskell
- Haskell Development Center
- Haskell Application Server
- Distributed Haskell (Cloud Haskell?)

FPComplete

The School of Haskell Story

- We wanted an early version of Education IDE
  - Interactive learning center
  - Lighter version of the IDE
  - Tools to create content
  - Tutorials embedded with editor with build and run
- The back end infrastructure was in place
  - Efficient and secure deployment system
  - Secure Content storage
  - Back up in GitHub

FPComplete

School of Haskell Goals

- Interactive on-line service
- Active code examples and exercises
- Free Accounts
- Support lots of users
- Low service cost per user
- Community contributed content
- Easily leverage existing content

School of Haskell Features

- User publishable content
- Support of standards (GitHub Markdown)
- Extensions for embedding active code
  - Focus on relevant code sections
  - Feature to hide sections of irrelevant code
  - Option to Show all code sections
  - Active code editor
  - Button to build and run
- Publisher toolkit for creating Markdown content

FPComplete

SoH Architecture

- Browser based client
- Web based server (front end)
- Isolated code execution containers (back end)
- Standard cloud services (services)

FPComplete

SoH Design

- Web based client / editor
  - Written in Javascript using codemirror
  - Implements user code editor
- Tutorials in Markdown
  - Render content as HTML
  - Overlay codemirror on code snippets
- Controls to
  - Show hidden code sections
  - Build and run code

FPComplete

Soh Front end services

- Code sent to front end for build and run
- Two way communication with application
  - Create “output window”
  - Send standard output to browser for display
  - Capture use input and send to application
- For Web tutorials
  - Launch an instance of web server
  - Code snippets served by web server
  - Frame for opening web server at a give URL

FPComplete

Markdown Extensions

- Active Code

  ```
  ```active Haskell Code ```
  ```

- Showing Code Snippets

  Mark visible code with “--show”

  ```
  ```active Haskell Code
  -- show
  Haskell Code Segment
  -- show
  Haskell Code ```
  ```

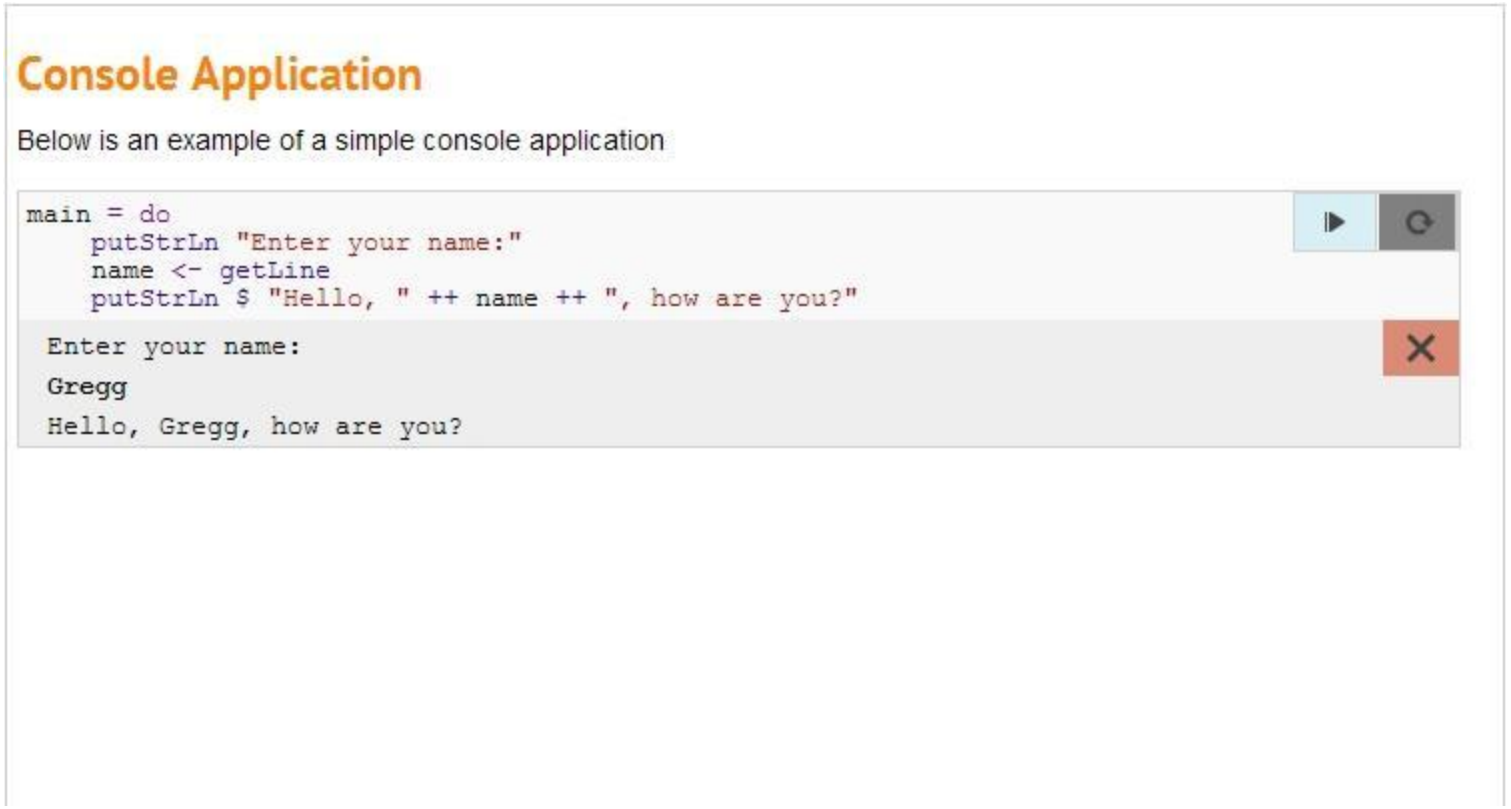
Example Console App

```
## Console Application

Below is an example of a simple console application


``` active haskell
main = do
    putStrLn "Enter your name:"
    name <- getLine
    putStrLn $ "Hello, " ++ name ++ ", how are you?"
```
```

Console App



Example Web App

```
## Sample Yesod App

This app creates a yesod server and
pages.

``` active haskell web
{-# LANGUAGE TypeFamilies,
QuasiQuotes, MultiParamTypeClasses,
       TemplateHaskell,
OverloadedStrings #-}
import Yesod

data MySite = MySite
instance Yesod MySite

mkYesod "MySite" [parseRoutes|
  /      HomeR  GET
  /about AboutR GET
|]

getHomeR = defaultLayout $ do
  [whamlet|
    Welcome to my web site!
    <br>
    <a href=@{AboutR}>About Me.
|]
  toWidget [cassius|
    body
      background-color: #048
      color: white
    a
      color: yellow
|]
getAboutR = defaultLayout [whamlet|
  Enough about me!
  <br> <a href=@{HomeR}>Back Home.
|]
main = warpEnv MySite
```
```

Showing Code Snippets

```
``` active haskell web
{-# LANGUAGE TypeFamilies,
QuasiQuotes, MultiParamTypeClasses,
       TemplateHaskell,
OverloadedStrings #-}
import Yesod

-- show
data MySite = MySite
instance Yesod MySite

mkYesod "MySite" [parseRoutes|
  /      HomeR  GET
  /about AboutR GET
|]

getHomeR = defaultLayout $ do
  [whamlet|
    Welcome to my web site!
    <br>
    <a href=@{AboutR}>About Me.
|]
  toWidget [cassius|
    body
      background-color: #048
      color: white
    a
      color: yellow
|]
getAboutR = defaultLayout [whamlet|
  Enough about me!
  <br> <a href=@{HomeR}>Back Home.
|]
main = warpEnv MySite
-- show
```
```

Example Snippet

```
data MySite = MySite

instance Yesod MySite

mkYesod "MySite" [parseRoutes|
  /       HomeR  GET
  /about AboutR GET
|]

getHomeR = defaultLayout $ do
  [whamlet|
    Welcome to my web site!
    <br>
    <a href=@{AboutR}>About Me.
  |]
  toWidget [cassius|
    body
      background-color: #048
      color: white
    a
      color: yellow
  |]

getAboutR = defaultLayout [whamlet|
  Enough about me!
  <br>
  <a href=@{HomeR}>Back Home.
|]

main = warpEnv MySite
```

Web App Output

Web Front End Design

- Written in Yesod
  - Whamlet, HTML, and Markdown for Web Pages
  - Engine for rendering Markdown
  - Cassius and CSS for Style Sheets
  - Fay for JavaScript (markdown can contain JS)
- Front end communicates via REST / JSON / AJAX
- All site static pages written in Markdown
- Markdown pages can contain HTML

FPComplete

Front End Functions

1. Serve Static Web Pages
2. Create Dynamic Pages (e.g. tutorial list)
3. Process Web Client Requests
4. Brokers back end services
5. Renders output from user applications

FPComplete

Infrastructure

- Isolation services (back end)
  - Creates and populates isolated runtimes
  - Serves requests for compile and execution
- Web servers (front end)
  - Manages virtual hosts and routes HTTP requests
  - Runs School of Haskell and user web services
- Cloud services (service infrastructure)
  - Storage (databases, and files)
  - Email and Messaging
  - Repository (GitHub, private git repos)
  - Logging and event management

FPComplete

Isolation Manager

- Manages isolation containers
  - Creates and populates containers
  - Launches build and execute processes
  - Sends stdout to front end service
- Isolation container options
  - LXC on a shared EC2 Instance
  - A private EC2 Instance
  - Created and destroyed on demand
  - Managed by the elastic load balancer

FPComplete

Front End Services

- Used for deploying services
- A collection of EC2 web instances
- Provides request routing
- Each web service assigned a host / port
- Virtual host manager responsible for routing
- Instances managed by front end EC2 balancer
- Instances communicate with cloud services

FPComplete

Cloud Services

- Can be any web based service
- SoH uses
  - S3,
  - GitHub and git
  - Postgres (Heroku)
  - SimpleDB
  - SES
  - Loggly

FPComplete

Add More Services

We will add more services based on

- Haskell Development Center requirements
- Individual requirements
- Customer demand for services

FPComplete

Next Steps

- Haskell Cloud Development Center
  - Code Snipped based intelligent editor
  - Guided development
  - Integrated help and learning
  - Build, test, and deploy
  - Repository integration
  - Vetted libraries from Stackage
  - Scalability
  - Cabal import and export

FPComplete

Next Steps

- Haskell Private Development Center
  - Same features as Cloud service
  - Deploy on public cloud or private VM
  - Available as a VM behind the firewall
  - Service is self managed
  - Updates from FP Complete

Next Steps

- Application services
  - Configuration and deployment management
  - Health monitoring and profiling
  - Alert and event management
  - Runtime Analytics and Metrics
  - Control Management
  - Exception processing
  - Event and log consolidation

FPComplete

Next Steps

- Interoperability
  - JVM
  - CLR
  - Improved C++
  - Java Application server support (e.g. Jboss)
  - Excel??

Next Steps

- Distributed Services
  - Cloud Haskell??
  - Distributed service management
  - Value Add distributed service Libraries
  - Failover and redundancy

FPComplete

Current Plans

- Alpha in April
- Public beta by mid year
- Some application services by end of year
- Some JVM support by end of year

FPComplete

That's it

Thank you for your time!!