d-fine

Igniting QuantLib on a Zeppelin

Andreas Pfadler, d-fine GmbH

QuantLib UserMeeting, Düsseldorf, 7.12.2016

Welcome Back!

- An early stage of this work has been presented at last year's user meeting
- As there has been a lot of positive encouragement by the audience, development has continued in the past year:
  - Large parts of the code have been rewritten
  - Proper build system (Maven) has been implemented
  - Code is published on GitHub (Apache License)
  - Docker images available
  - Integration of Apache Zeppelin
- Please note: This is a **personal** project of mine. As such, it has been created in my free time. Neither I nor my employer do provide any sort of guarantees, warranty, support, etc. You are, however, free to use the source code according to the terms of the Apache License.

Agenda

- Short Recap
  - Motivation for this talk
  - Architectural ingredients
- Live Demo
- Outlook

Motivation

- There exist a number of commercial closed source platforms and in-house systems which in some way combine
  - Analytics, i.e. Pricing Libraries, Exposure Engines, …
  - Grid computing frameworks
  - Grid wide caching / in memory computing
  - HTTP/Rest interfaces and HTML5 GUIs
  - Excel Integration
  - Workflows for product/model development and deployment
- In this talk we show how we can build similar systems using open source software only.
- Our main ingredients are
  - QuantLib with Swig Java bindings
  - Apache Ignite
  - Jetty
  - Scala
  - AngularJS
  - Apache Zeppelin

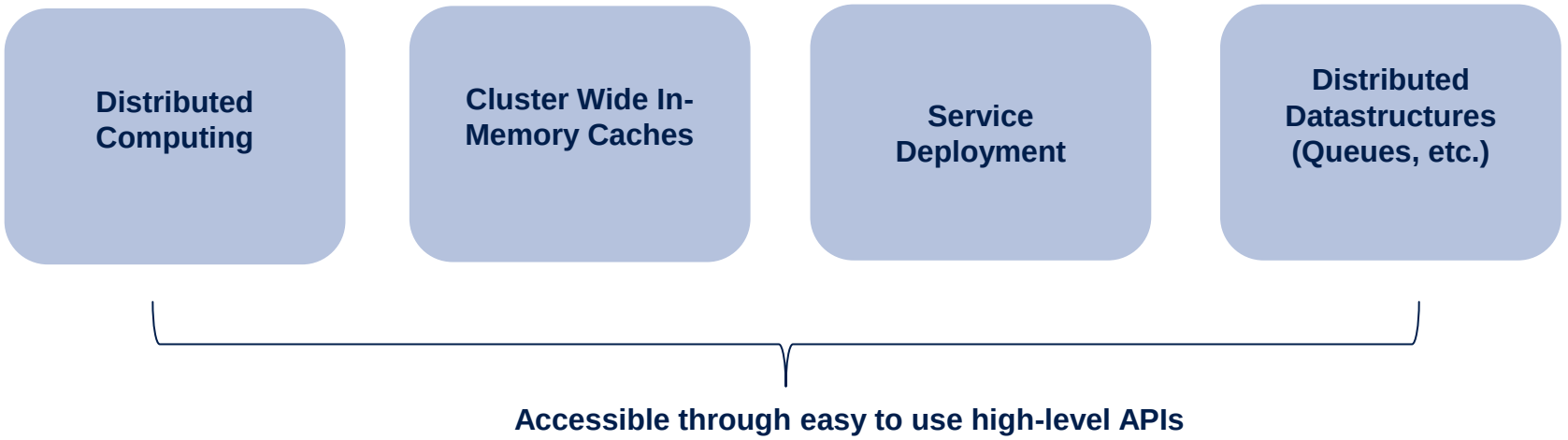
d-fine

Basic Ingredients

Apache Ignite (1)

- Originally developed by Grid Gain Systems
- Recently (2015) promoted to a top level project of the Apache Software Foundation
- Can be thought of as a kind of **distributed in memory data fabric.**
- Based on Java. Configurable using Spring, sometimes seen as a „competitor“ to Apache Spark.

Main Features:



Apache Ignite (2)

» **Example 1: distributed closure** (see Ignite documentation)

```
IgniteCompute compute = ignite.compute();

// Execute closure on all cluster nodes.
Collection<Integer> res = compute.apply( String::length, Arrays.asList("How many characters".split(" ")) );

// Add all the word lengths received from cluster nodes.

 int total = res.stream().mapToInt(Integer::intValue).sum();
```

» **Example 2: Query grid-wide cache using a predicate** (see Ignite Documentation)

```
IgniteCache<Long, Person> cache = ignite.cache("mycache");

// Find only persons earning more than 1,000.
 try (QueryCursor cursor = cache.query(new ScanQuery((k, p) -> p.getSalary() > 1000))
{
        for (Person p : cursor)
             System.out.println(p.toString());
}
```

d-fine

REST/HTML5 Interface: Jetty and AngularJS

Jetty

- Traditional web apps are deployed as packages on some application server
- If your application is already implemented in some sort of daemon process anyway and you don‘t want to run a full application server, why not directly embed a http server?
- Standard option in the Java world: **Jetty**
- Features easy integration (Maven packages) and allows for running servlets or even fully fledged applications packaged in a .war file.

AngularJS

- Maintained mostly by Google
- Targeted at single page apps based on MVC pattern
- Features custom HTML5 directives and bidirectional data binding
- Makes JS development less painful…

QuantLib and Scala Integration

QuantLib / JNI / Thread-Safety

- Thread safety: Since we will embed QuantLib through JNI and run several Java threads in parallel across a number of Ignite nodes, this is of paramount importance.
- Need to make use of Klaus Spanderen‘s implementation of the thread safe observer pattern (https://hpcquantlib.wordpress.com/2013/07/26/multi-threading-and-quantlib/) – already part of QL since 1.7.
- Have to make sure that we have one session per thread.

Scala Integration

- Nice to have: Generic Scripting facility, e.g. to define payoffs or products
- Also nice to have: Offer users a way to quickly achieve results using an embedded scripting facility similar to IPython notebooks, etc.
- Thus we integrate a Scala interpreter into our system (although Jython would be equally well suited…)
- Inspired again by https://hpcquantlib.wordpress.com/2011/09/01/using-scala-for-payoff-scripting/.

Apache Zeppelin

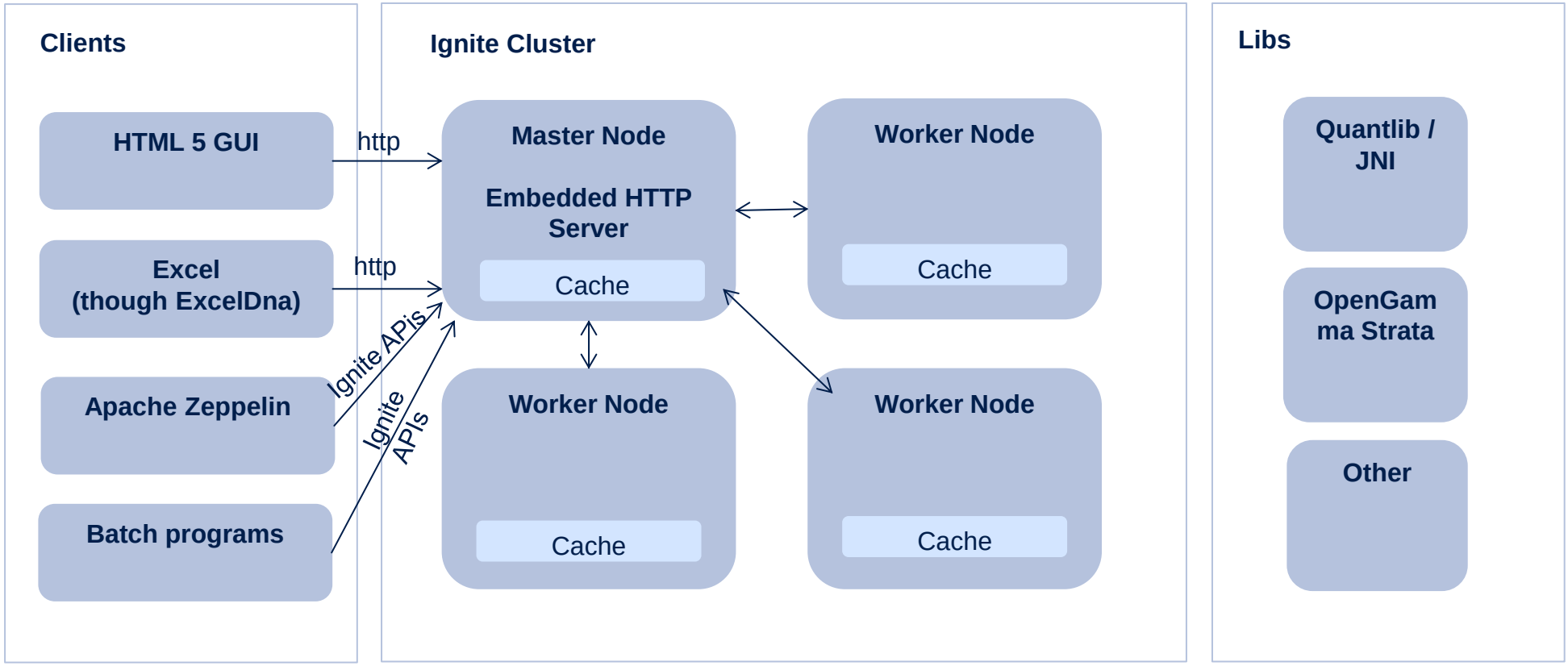
- One reason for the recent growth Python‘s popularity consists of course in the famous IPython notebook. It would be nice to have something similar in our system, especially for rapid prototyping or ad-Hoc analytics / reports

- This why we connect **Apache Zeppelin**: It offers a **generic web-based, IPython/Mathematica style notebooks.**

- Also: It is shipped with a builtin „Ignite-Interpreter“. This is basically an embedded Scala-Interpreter with an Ignite context object, that may be used to access all of the grid computing and grid cache facilities offered by Ignite.

- Moreover, one can also integrate other „Intepreters“, which may for instance be used to query existing databases or submit jobs to a Spark Cluster.

The Big Picture



Live Demo

Outlook

A lot remains to be done

- The architecture provides a solid foundation for distributed pricing combined with an easy to use cluster wide caching mechanism
- However, for concrete applications a lot remains to be inspected closer or worked on in the future:
  - Overall stability and performance / Integration into production environments
  - High-Level Market Data API on top of Ignite cache and integration of live market data feeds
  - Security considerations
  - Flexible and easy to use ways for defining new products and models beyond the simple examples shown here
  - Clear workflows for quantitative development und production deployment

Key Message

- There are great open source projects that help you building sophisticated and easy to use pricing platforms
- Don‘t start developing from scratch and keep an open mind with regards to what‘s out there in terms of (not only finance related) open source software

Code and Docker Images

Source code

- Main source code on github: **apfadler/quil-src**
- Build requirements:
  - JDK 8
  - Maven
  - QuantlibJNI.dll/.so for QL 1.7. (Talk to me if you need binaries)
  - For Windows: git bash or Cygwin shell (the latter has not been tested yet)

Docker images

- apfadler/quantlib-quil-server, apfadler/quantlib-zeppelin on dockerhub
- Simplest way to use them: See **apfadler/quil-docker** on github.

Dr. Andreas Pfadler
Manager
andreas.pfadler@d-fine.de
Tel +49 (0)69 90737 0
Mobil +49 (0)162 2630029

d-fine GmbH

Frankfurt
München
London
Wien
Zürich

Zentrale

d-fine GmbH
Opernplatz 2
D-60313 Frankfurt/Main

Tel +49 69 90737-0
Fax +49 69 90737-200

www.d-fine.com


d—fine