Building High Performance Microservices with Apache Thrift

Rethinking service APIs in a Cloud Native environment
Presenters

• **Randy** Abernethy
  • ra@apache.org, randy.abernethy@rx-m.com
  • Apache Thrift PMC
  • CNCF member
  • RX-M Cloud Native Consulting Partner

• **Nobuaki** Sukegawa
  • nsuke@apache.org
  • Apache Thrift PMC
  • Bloomberg Engineer
What is Cloud Native?

- Microservice Oriented
- Container Packaged
- Dynamically Orchestrated
Problems Cloud Native Solutions Can Solve:

• Extreme horizontal scale
• Increased server density
• Granular scaling
• Improved and explicit modularity
• Support for aspirational development processes
  • CI/CD
  • Agile development
  • Everything as code
• Support for rapid adoption of new technologies
  • Time to Innovation/Market
Cloud Native Adoption

- Docker hub has seen 390,000% image pull growth since 2014
- K8s has seen 567% growth in commits just over a year
- Google starts over 3,000 containers per second in their Borg/Omega environment
- Pokemon Go is a 30,000 container cloud native application running on Google Container Engine

### Kubernetes Commits

<table>
<thead>
<tr>
<th>month</th>
<th>commits</th>
</tr>
</thead>
<tbody>
<tr>
<td>2016-01</td>
<td>199</td>
</tr>
<tr>
<td>2016-02</td>
<td>161</td>
</tr>
<tr>
<td>2016-03</td>
<td>229</td>
</tr>
<tr>
<td>2016-04</td>
<td>180</td>
</tr>
<tr>
<td>2016-05</td>
<td>374</td>
</tr>
<tr>
<td>2016-06</td>
<td>309</td>
</tr>
<tr>
<td>2016-07</td>
<td>331</td>
</tr>
<tr>
<td>2016-08</td>
<td>221</td>
</tr>
<tr>
<td>2016-09</td>
<td>362</td>
</tr>
<tr>
<td>2016-10</td>
<td>551</td>
</tr>
<tr>
<td>2016-11</td>
<td>632</td>
</tr>
<tr>
<td>2016-12</td>
<td>919</td>
</tr>
<tr>
<td>2017-01</td>
<td>736</td>
</tr>
<tr>
<td>2017-02</td>
<td>877</td>
</tr>
<tr>
<td>2017-03</td>
<td>1177</td>
</tr>
<tr>
<td>2017-04</td>
<td>1130</td>
</tr>
</tbody>
</table>

Source: Docker Blog
Contrasting Containers with VMs

- **VM A**: Hypervisor
  - OS
  - VM A
  - App A

- **VM B**: Hypervisor
  - OS
  - VM B
  - App B

- **Container A**: OS
  - App A

- **Container B**: OS
  - App B

H/W
Challenges created by a microservice approach

- Explosion in the number of service instances to manage
- Extreme need for reliable deployment
- Dramatically different debugging and monitoring models
- Increased pressure on networks to exchange procedure calls

Netflix Microservice “Death Star” Model
**Modern RPC**

- What is modern RPC?
  - Cross platform
  - Polyglot
  - Evolvable
  - Fast

- Monoliths are internally composed of modules which call each other through exposed functions/methods

- This model is easy to translate to RPC style microservices

- The largest Internet Scale firms have all adopted Modern RPC solutions internally to improve service performance
  - Google – ProtoBuf/Stubby (now moving from Stubby to gRPC)
  - Facebook – Thrift
  - Twitter – Thrift/Scrooge/Finagle

---

1980 - Bruce Jay Nelson is credited with inventing the term **RPC** in early ARPANET documents
- The idea of treating network operations as procedure calls

1981 - Xerox Courier possibly the first commercial RPC system

1984 - Sun RPC (now Open Network Computing [ONC+] RPC, RFC 5531)

1991 - CORBA – Common Object Request Broker Architecture
- The CORBA specification defines an ORB through which an application interacts with objects
- Applications typically initialize the ORB and accesses an internal Object Adapter, which maintains things like reference counting, object (and reference) instantiation policies, and object lifetime policies
- General Inter-ORB Protocol (GIOP) is the abstract protocol by which object request brokers (ORBs) communicate
- Internet InterORB Protocol (IIOP) is an implementation of the GIOP for use over the Internet, and provides a mapping between GIOP messages and the TCP/IP layer

1993 - DCE RPC – An open (designed by committee) RPC solution integrated with the Distributed Computing Environment
- Packaged with a distributed file system, network information system and other platform elements

1994 - MS RPC (a flavor of DCE RPC and the basis for DCOM)

1994 - Java RMI – A Java API that performs the object-oriented equivalent of remote procedure calls (RPC), with support for direct transfer of serialized Java objects and distributed garbage collection
- RMI-IIOP implements the RMI interface over CORBA
- Third party RMI implementations and wrappers are prevalent (e.g. Spring RMI)

1998 - SOAP (Simple Object Access Protocol) specifies a way to perform RPC using XML over HTTP or Simple Mail Transfer Protocol (SMTP) for message negotiation and transmission

2001 - Google Protocol Buffers – developed at Google to glue their servers together and interoperate between their three official languages (C++/Java/Python), JavaScript and others have since been added, used as a serialization scheme for custom RPC systems

2006 - Apache Thrift – developed at Facebook to solve REST performance problems and to glue their servers together across many languages
- The basis for Twitter Finagle, a cornerstone of the Twitter platform

2008 - Apache Avro is a serialization framework designed to package the serialization schema with the data serialized, packaged with Hadoop

2015 - Google gRPC announced as an RPC framework operating over http/2 using protocol buffers for serialization

2017 - Google contributes gRPC to CNCF
Fast does not have to be hard

• To create an Apache Thrift service, simply:
  • Define it in IDL
  • Generate client stubs in your choice of languages
  • Generate a server stub and wire it to your implementation
  • Use a prebuilt Apache Thrift server shell to implement the service

Apache Thrift supplies all of the components necessary to turn a set of functions into a network based microservice accessible from a range of platforms and languages.
Isn’t REST fast enough?

• For public, widely consumed APIs, REST is very good, leveraging the infrastructure of the web.

• For internal, high performance APIs, REST, HTTP and JSON text serialization can be slow and there’s no “web infra” to leverage.

• The chart at right shows seconds required for the same client on the same computer to call the same local service 1mm times.

• Each bar, uses a different tech stack to implement the service.

Service Performance Comparison

- SOAP (JAX-WS, Tomcat 7, HTTP, XML
- REST (JAX-RS/Jersey 2), Tomcat 7, HTTP, JSON
- Apache Thrift, Tomcat 7, HTTP, JSON
- Apache Thrift, TCP, JSON
- Apache Thrift, TCP, Compact
Thrift Language Comparison

Thrift Local Loopback Test
Time to complete 1mm calls
Client (Cli->) to server (->Svr)
Performance in the backend

- Thrift
  - Compact Protocol
  - TCP
- gRPC
  - ProtoBuf
  - HTTP/2
    - POST
Performance over the Internet

- The world wide web is the largest distributed system mankind has ever created
- Systems leveraging the protocols of the Web (http/http/2) gain many benefits at little or no cost
  - Massively distributed caches
  - Security appliances/technologies
  - Gateways
  - Loadbalancers
  - Etc.
- REST and to some degree gRPC and Thrift over http reap many of these benefits

Fielding, 2000
Imagine we need to build a service that tracks OPEN SOURCE PROJECTS

Demo

Part I: Creating a Thrift microservice, containerizing it, orchestrating it
Part II: Evolving the service and rolling it out without breaking compatibility

https://github.com/RX-M/api-bench
Key Features of Apache Thrift

- **Servers and Serialization** – a complete serialization and service solution in tree
- **Modularity** – pluggable serialization protocols and transports with a range of provided implementations
- **Performance** – light weight, scalable servers with fast and efficient serialization
- **Reach** – support for an impressive range of languages, protocols and platforms
- **Rich IDL** – language independent support for expressive type and service abstractions
- **Flexibility** – integrated type and service evolution features
- **Community Driven Open Source** – Apache Software Foundation hosted and community managed
Thank you!

Questions?