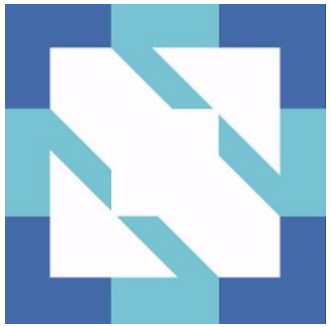
# Building High Performance Microservices with Apache Thrift

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

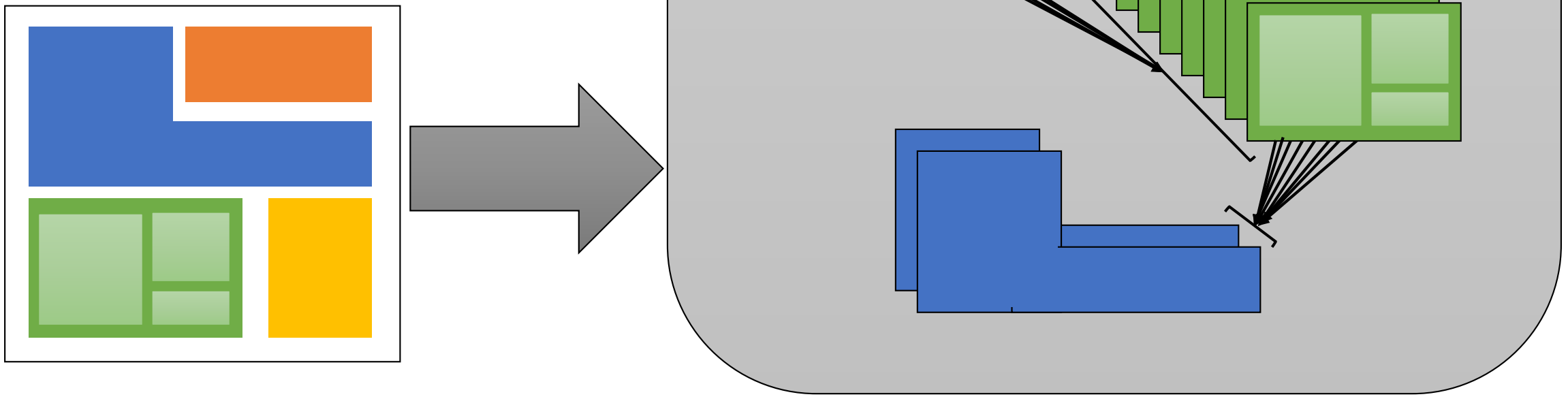
RX-M Cloud Native Consulting

- **Nobuaki** Sukegawa
  - nsuke@apache.org
  - Apache Thrift  PMC
  - Bloomberg Engineer

Bloomberg

# What is Cloud Native?

- Microservice Oriented
- Container Packaged
- Dynamically Orchestrated



# Problems Cloud Native Solutions Can Solve:

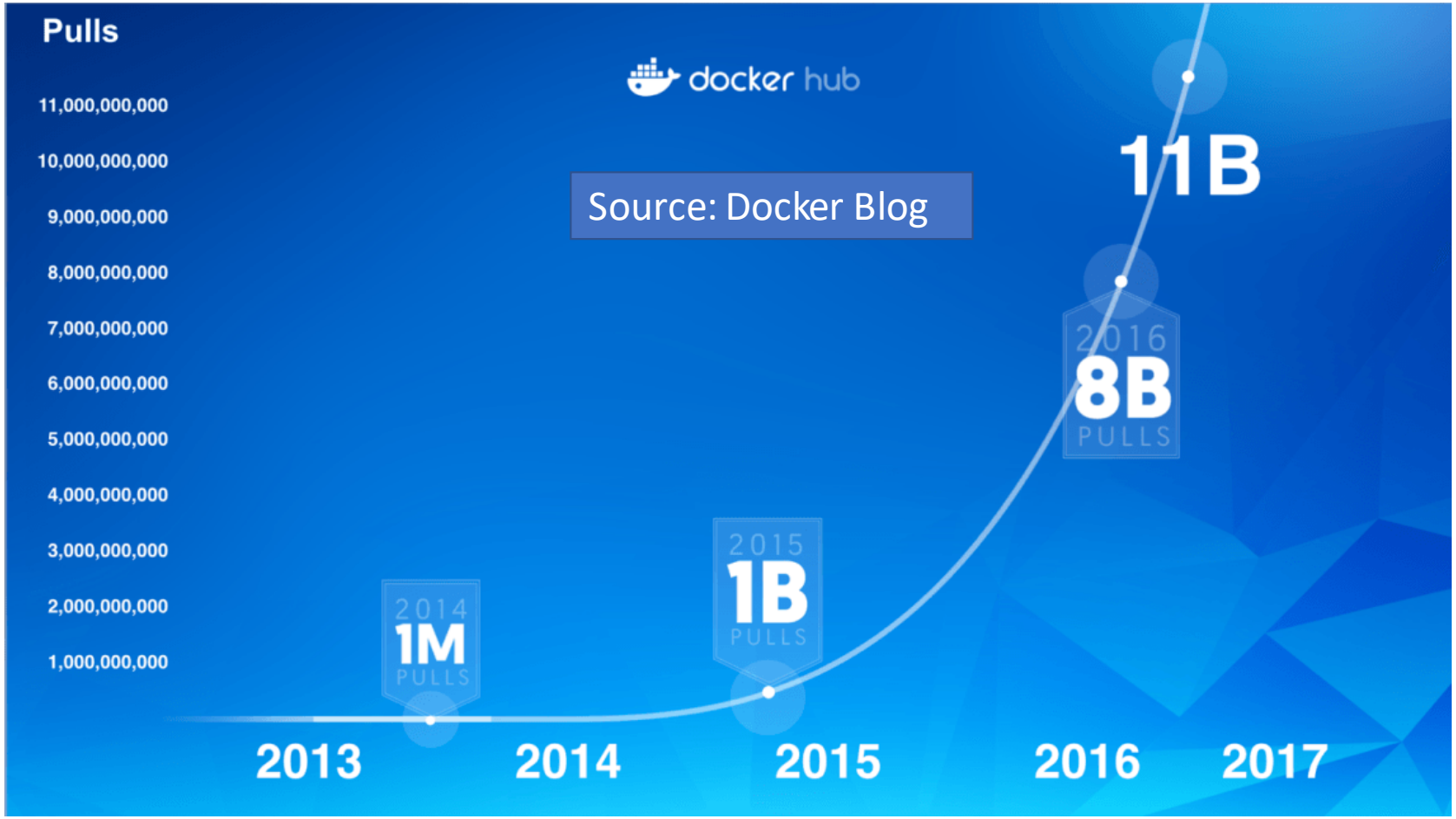
- Extreme horizontal scale
- Increased server density
- Granular scaling
- Improved and explicit modularity
- Support for aspirational development processes
  - CI/CD
  - Agile development
  - Everything as code
- Support for rapid adoption of new technologies
- **Time to Innovation/Market**

# Cloud Native Adoption

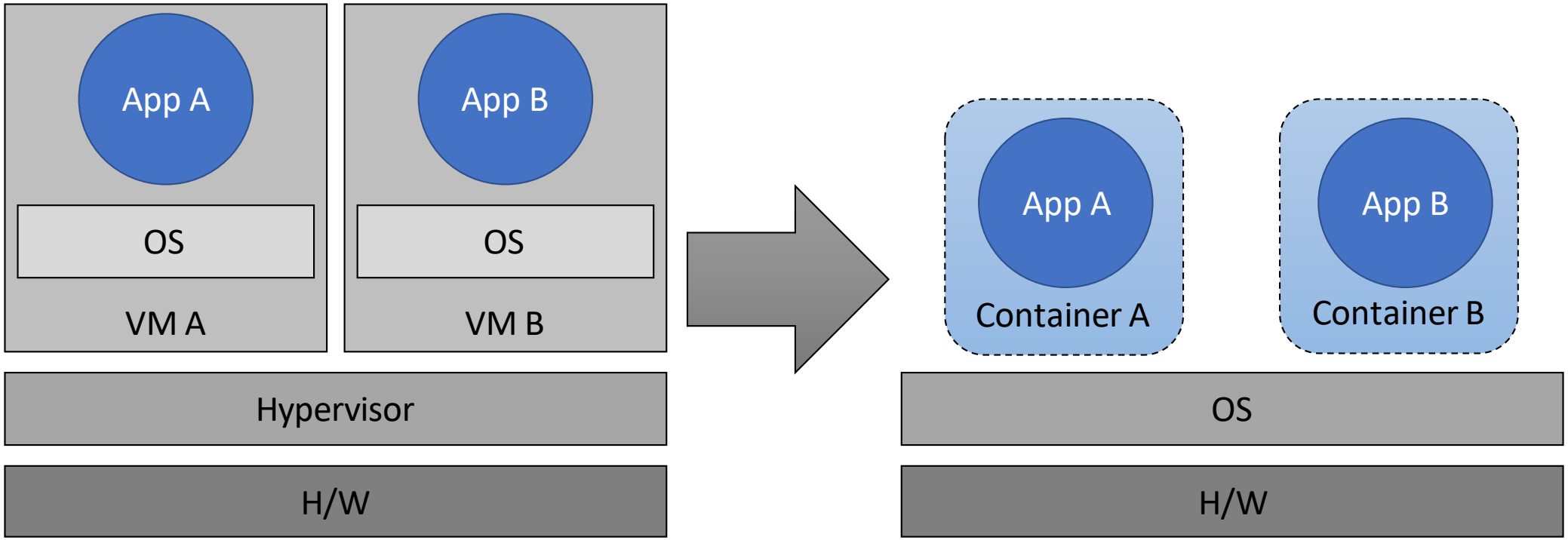
- Docker hub has seen 390,000% image pull growth since 2014
- K8s has seen 567% growth in commits just over a year
- Google starts over 3,000 containers per second in their Borg/Omega environment
- Pokemon Go is a 30,000 container cloud native application running on Google Container Engine

Kubernetes Commits

| month | commits |
|---|---|
| 2016-01 | 199 |
| 2016-02 | 161 |
| 2016-03 | 229 |
| 2016-04 | 180 |
| 2016-05 | 374 |
| 2016-06 | 309 |
| 2016-07 | 331 |
| 2016-08 | 221 |
| 2016-09 | 362 |
| 2016-10 | 551 |
| 2016-11 | 632 |
| 2016-12 | 919 |
| 2017-01 | 736 |
| 2017-02 | 877 |
| 2017-03 | 1177 |
| 2017-04 | 1130 |



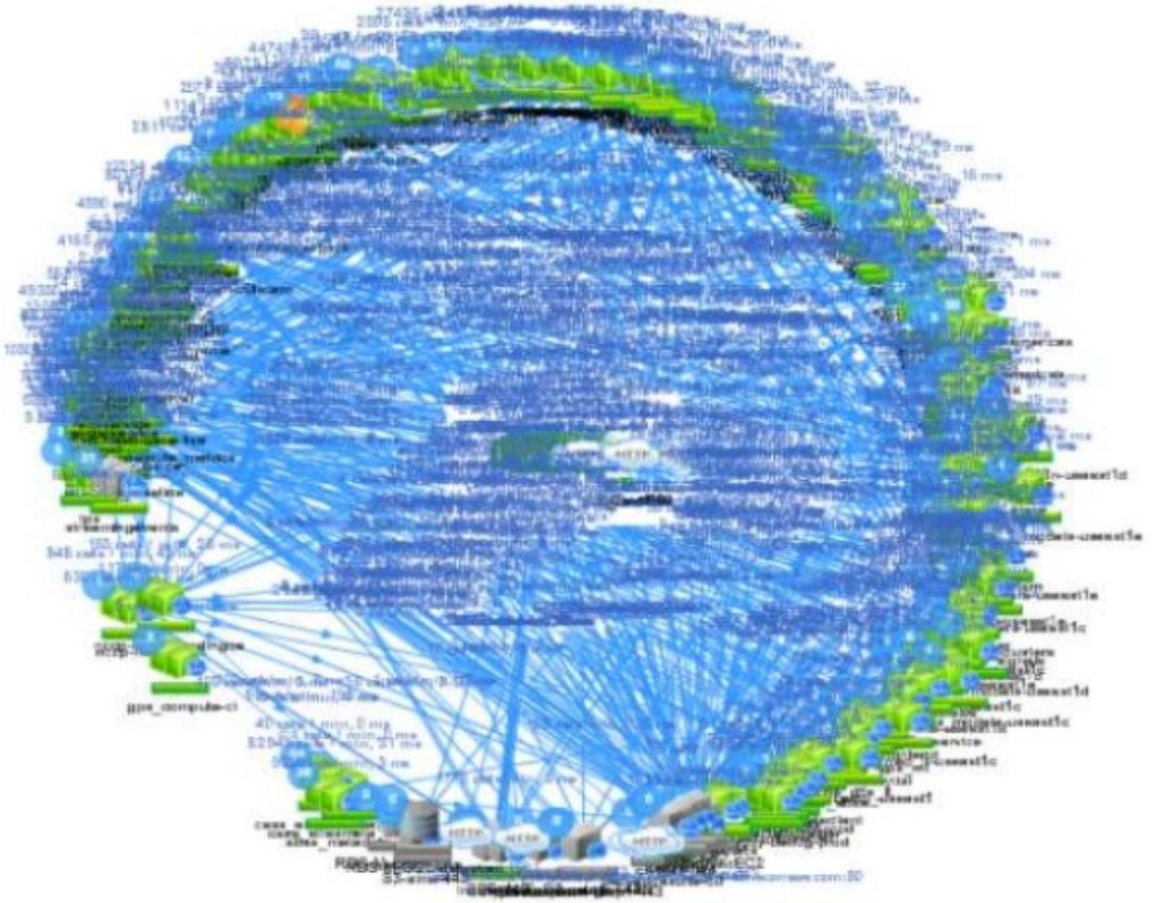
# Contrasting Containers with VMs



# 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

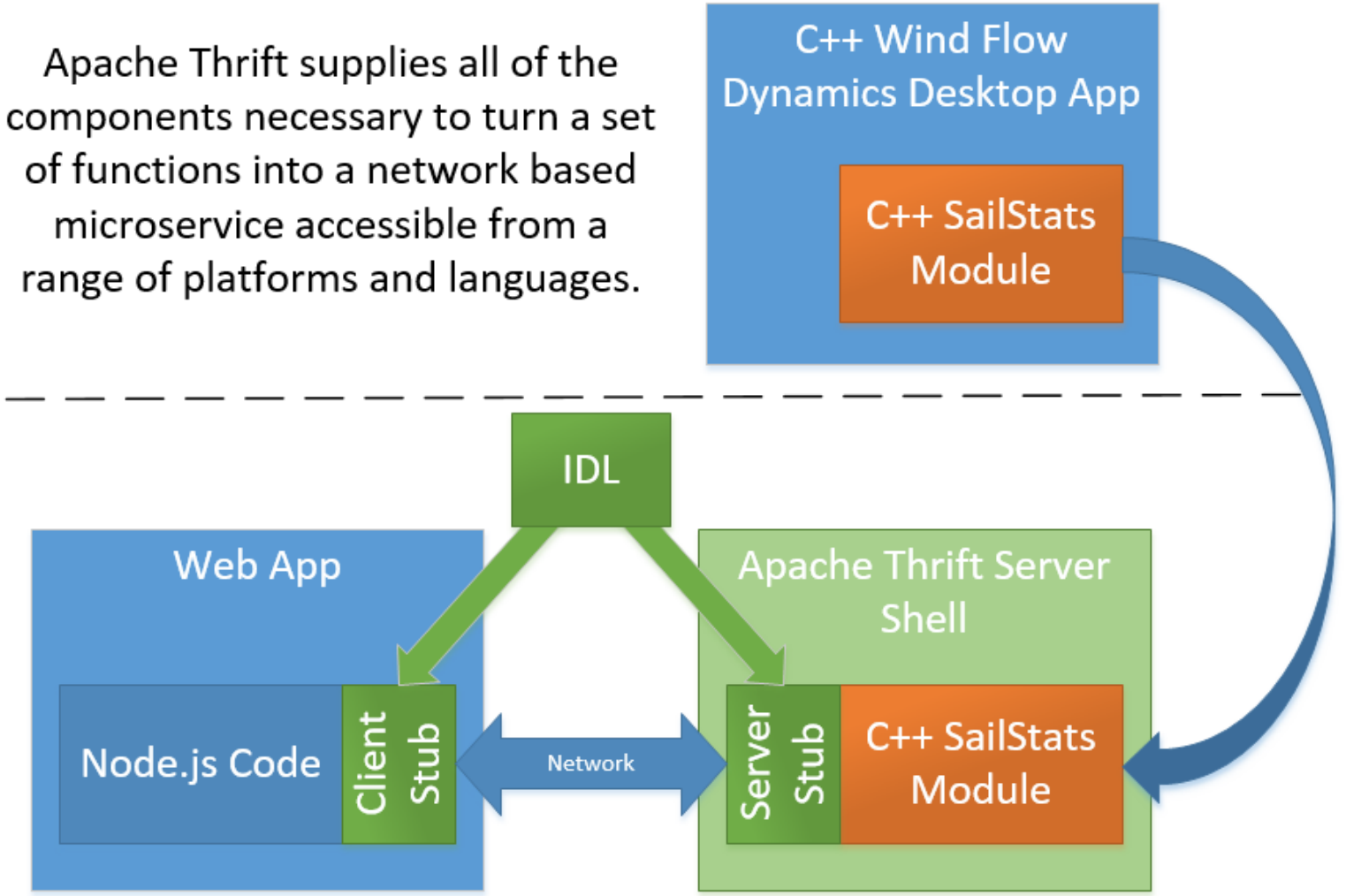
2017 - Google contributes gRPC to CNCF

# Fast does not have to be hard

```
service SailStats {
     double get_sailor_rating(1: string sailor_name)
     double get_team_rating(1: string team_name)
     double get_boat_rating(1: i64 boat_serial_number)
     list<string> get_sailors_on_team(1: string team_name)
     list<string> get_sailors_rated_between(1: double min_rating,
                                            2: double max_rating)
     string get_team_captain(1: string team_name)
}
```

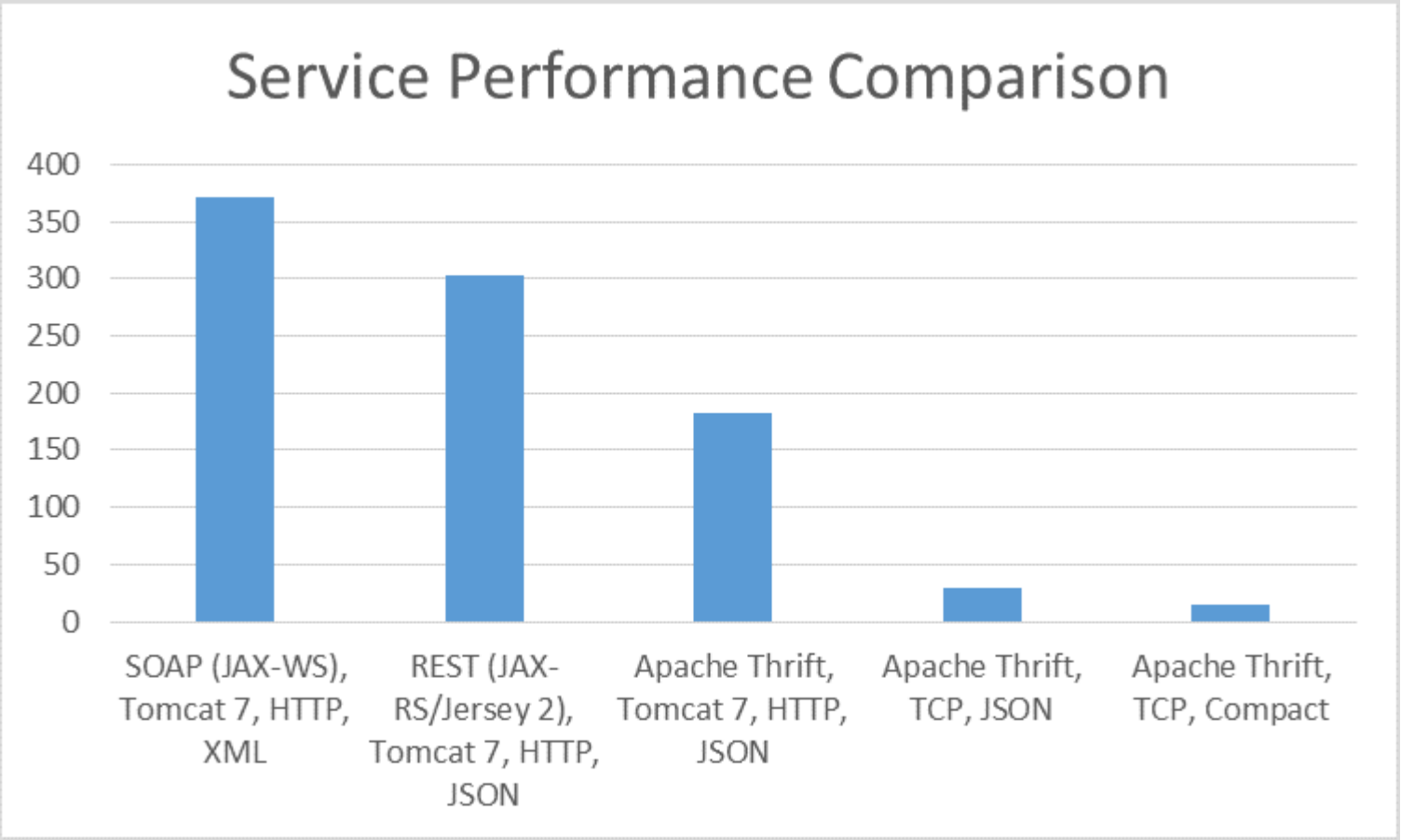
- To create an Apache Thrift service, simply:
  - Define it in IDL
  - Generate client stubs in your choice of languages
  - Generate a server stub and wire it to your implementation
  - Use a prebuilt Apache Thrift server shell to implement the service

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.

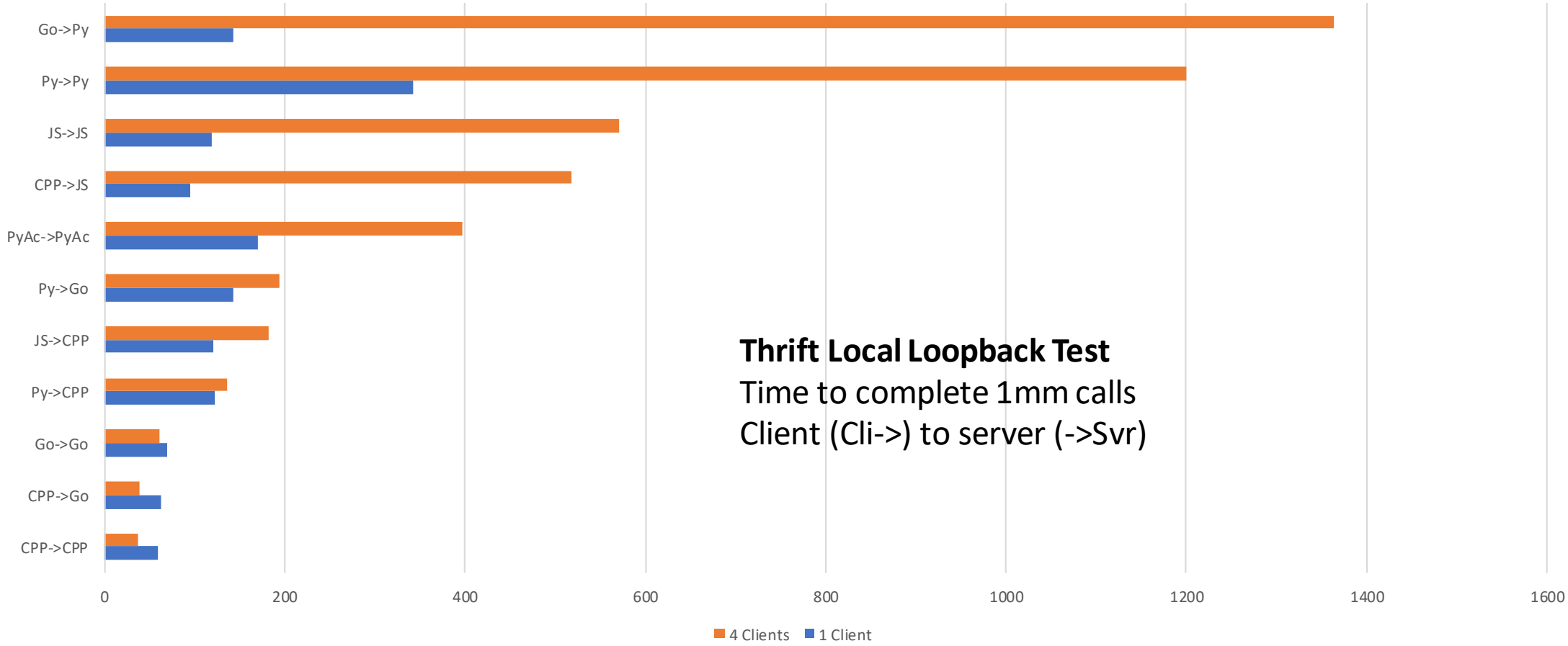


# Isn't REST fast enough?

- 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

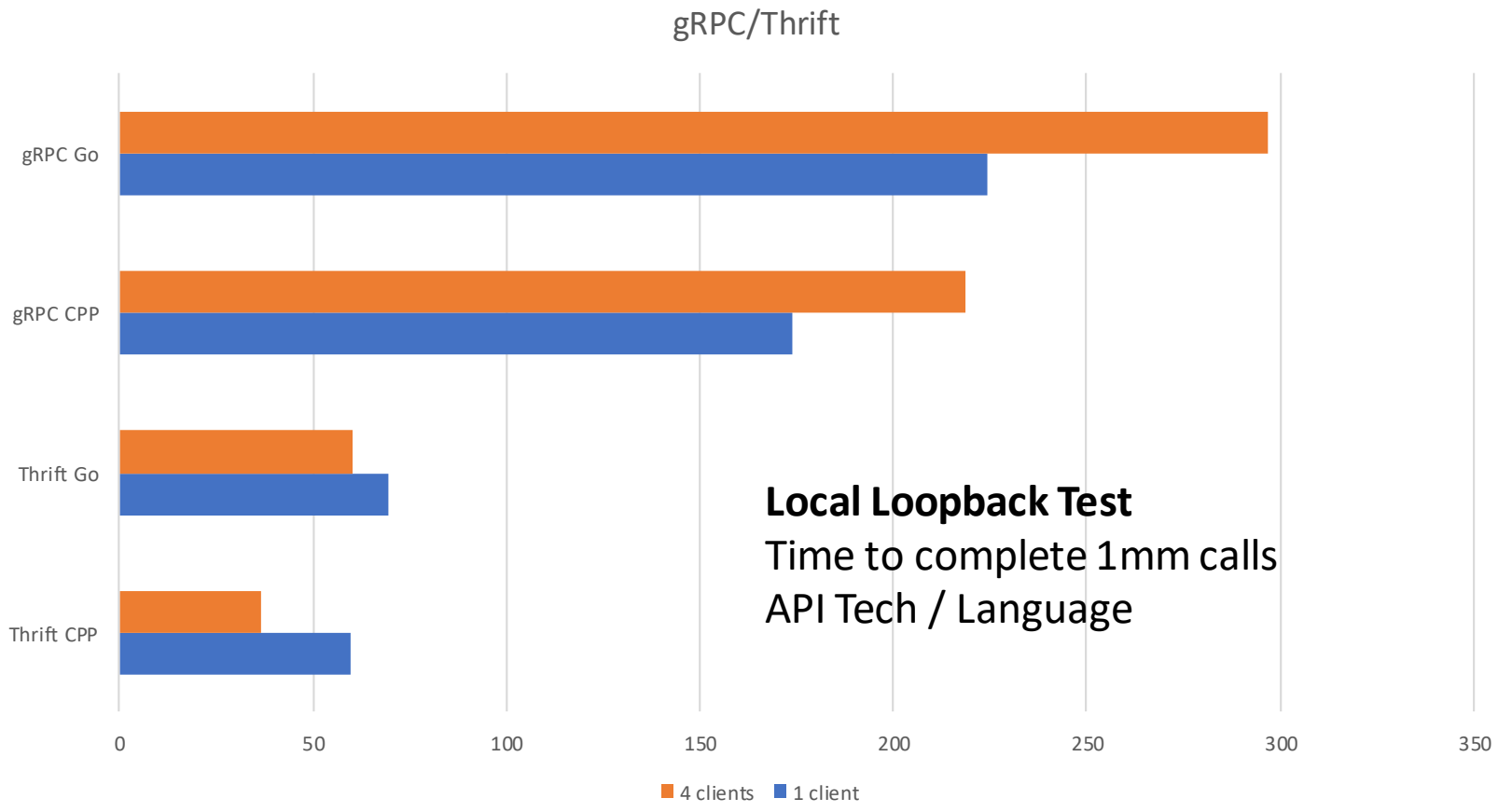


# Thrift Language Comparison



# 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

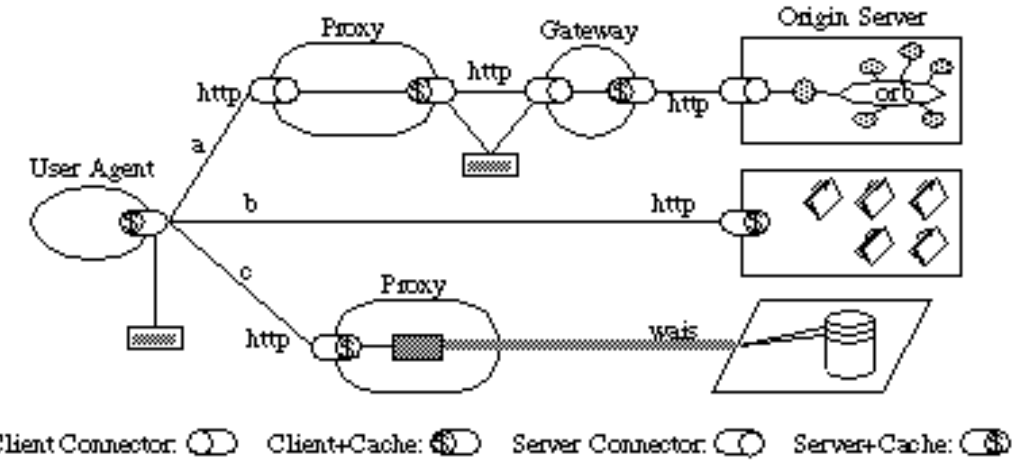



Figure 5-10. Process View of a REST-based Architecture

A user agent is portrayed in the midst of three parallel interactions: a, b, and c. The interactions were not satisfied by the user agent's client connector cache, so each request has been routed to the resource origin according to the properties of each resource identifier and the configuration of the client connector. Request (a) has been sent to a local proxy, which in turn accesses a caching gateway found by DNS lookup, which forwards the request on to be satisfied by an origin server whose internal resources are defined by an encapsulated object request broker architecture. Request (b) is sent directly to an origin server, which is able to satisfy the request from its own cache. Request (c) is sent to a proxy that is capable of directly accessing WAIS, an information service that is separate from the Web architecture, and translating the WAIS response into a format recognized by the generic connector interface. Each component is only aware of the interaction with their own client or server connectors; the overall process topology is an artifact of our view.

*Fielding, 2000*

https://github.com/RX-M/api-bench

Imagine we need to build a service that tracks

**OPEN SOURCE PROJECTS**

```
namespace * OpenSourceProjects

struct Date {
    1: i16 year
    2: i16 month
    3: i16 day
}


struct Project {
    1: string name       //Proper project name
    2: string host       //Hosting/control foundation or company
    3: Date inception    //Date project was open sourced
}

struct CreateResult {
    1: i16 code
    2: string message
}

service Projects {
    Project get(1: string name)
    CreateResult create(1: Project proj)
}
```

# Demo

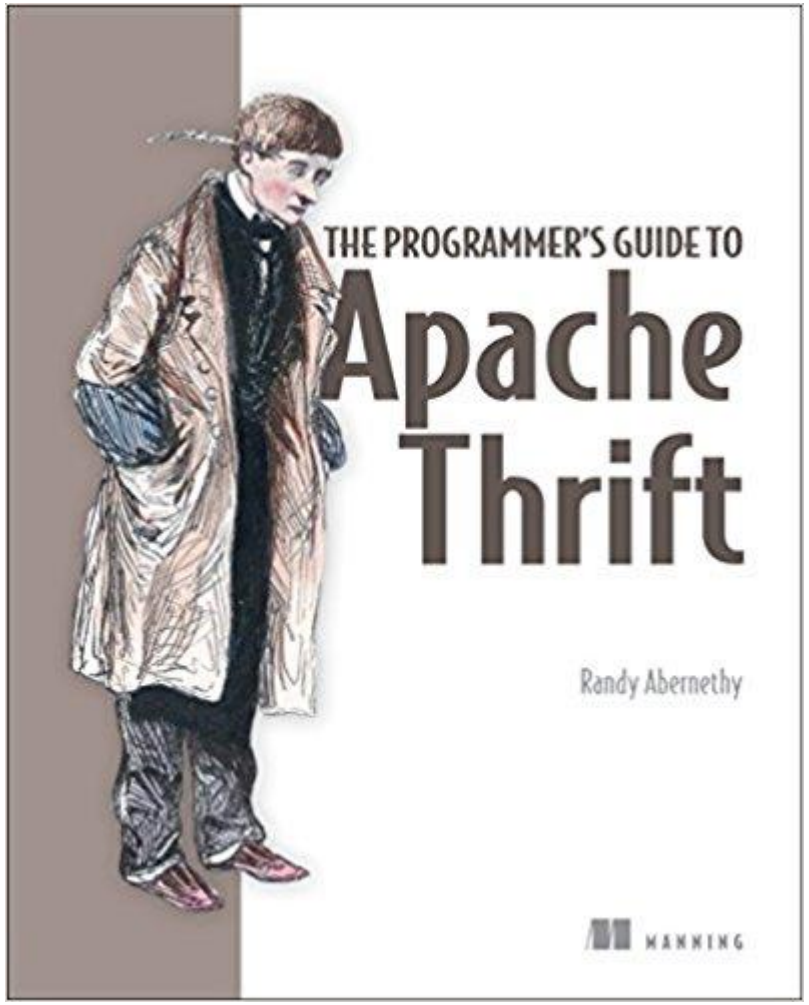
Part I: Creating a Thrift microservice, containerizing it, orchestrating it

Part II: Evolving the service and rolling it out without breaking compatibility

# Apache Thrift Take Away

- 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

*39% discount code: abernethydz*
*Good at Manning.com*



# Thank you!

Questions?