# QCon SF by InfoQ

- **Bleeding-edge Software Developer Conference for the Enterprise**
- **18 editorial tracks across 3 days**
- **140+ practitioner speakers**

<table>
<thead>
<tr>
<th>Monday, 5 November</th>
<th>Tuesday, 6 November</th>
<th>Wednesday, 7 November</th>
</tr>
</thead>
<tbody>
<tr>
<td>Microservices / Serverless Patterns &amp; Practices</td>
<td>Architectures You’ve Always Wondered About</td>
<td>Applied AI &amp; Machine Learning</td>
</tr>
<tr>
<td>Evolving, observing, persisting, and building modern microservices</td>
<td>Next-gen architectures from the most admired companies in software, such as Netflix, Google, Facebook, Twitter, &amp; more</td>
<td>Applied machine learning lessons for SWEs, including tech around TensorFlow, TPUs, Keras, PyTorch, &amp; more</td>
</tr>
<tr>
<td>Practices of DevOps &amp; Lean Thinking</td>
<td>21st Century Languages</td>
<td>Production Readiness: Building Resilient Systems</td>
</tr>
<tr>
<td>Practical approaches using DevOps &amp; Lean Thinking</td>
<td>Lessons learned from languages like Rust, Go-lang, Swift, Kotlin, and more.</td>
<td>More than just building software, building deployable production ready software</td>
</tr>
</tbody>
</table>
Tracks

Modern CS in the Real World
- Thoughts pushing software forward, including consensus, CRDT's, formal methods, & probabilistic programming

Bare Knuckle Performance
- Killing latency and getting the most out of your hardware

Security: Lessons Attacking & Defending
- Security from the defender's AND the attacker's point of view

Modern Operating Systems
- Applied, practical, & real-world deep-dive into industry adoption of OS, containers and virtualization, including Linux on Windows, ...

Socially Conscious Software
- Building socially responsible software that protects users privacy & safety

Future of Human Computer Interaction
- IoT, voice, mobile: Interfaces pushing the boundary of what we consider to be the interface

Optimizing You: Human Skills for Individuals
- Better teams start with a better self. Learn practical skills for IC

Delivering on the Promise of Containers
- Runtime containers, libraries, and services that power microservices

Enterprise Languages
- Workhorse languages found in modern enterprises. Expect Java, .NET, & Node in this track
“Accelerating the software side of human technological progress”

“Stay ahead of the adoption curve”
References

Java

- https://www.infoq.com/java

QCon

- https://www.infoq.com/qcon/
- https://qconsf.com/volunteers
Netflix Play API
Why we built an Evolutionary Architecture

Suudhan Rangarajan (@suudhan)
Senior Software Engineer
Identity
Type 1/2 Decisions
Evolvability
Start with WHY: Ask why your service exists

API Identity: Deliver Acquisition, Discovery and Playback functions with high availability
Single Responsibility Principle: Be wary of multiple-identities rolled up into a single service.
Play API Identity: **Orchestrate Playback Lifecycle with stable abstractions**

![Diagram](image-url)

- Devices
- API Proxy Service
- Play API
- Decide best playback experience
- Authorize playback experience
- Track events to measure playback experience
Guiding Principle: We believe in a simple singular identity for our services. The identity relates to and complements the identities of the company, organization, team and its peer services.
“Some decisions are consequential and irreversible or nearly irreversible – one-way doors – and these decisions must be made methodically, carefully, slowly, with great deliberation and consultation [...] We can call these Type 1 decisions…”

“...But most decisions aren’t like that – they are changeable, reversible – they’re two-way doors. If you’ve made a suboptimal Type 2 decision, you don’t have to live with the consequences for that long [...] Type 2 decisions can and should be made quickly by high judgment individuals or small groups.”

Three Type 1 Decisions to Consider

Quote from Jeff Bezos
Two types of Shared Libraries

- Utilities
- cache
- Metrics

Shared Libraries with common functions

Client Libraries used for inter-service communications

Binary coupling => Distributed Monolith

“The evils of too much coupling between services are far worse than the problems caused by code duplication”

- Sam Newman (Building Microservices)
“Operational Coupling” might be an ok choice, if some services/teams are not yet ready to own and operate a highly available service.

Clients with heavy Fallbacks

Operational Coupling impacts Availability

Many of the client libraries had the potential to bring down the API Service
## Requirements

<table>
<thead>
<tr>
<th>Operationally “thin” Clients</th>
<th>No or limited shared libraries</th>
</tr>
</thead>
<tbody>
<tr>
<td>Auto-generated clients for Polyglot support</td>
<td>Bi-Directional Communication</td>
</tr>
</tbody>
</table>

## REST vs RPC

- At Netflix, most use-cases were modelled as Request/Response
  - REST was a simple and easy way of communicating between services; so choice of REST was more incidental rather than intentional
- Most of the services were not following RESTful principles.
  - The URL didn’t represent a unique resource, instead the parameters passed in the call determined the response - effectively made them a RPC call
- So we were agnostic to REST vs RPC as long as it meets our requirements
1) Operationally Coupled Clients
2) High Binary Coupling
3) Only Java
4) Unidirectional communication

1) Minimal Operational Coupling
2) Limited Binary Coupling
3) Beyond Java
4) Beyond Request/Response
Type 1 Decision: Appropriate Coupling

Consider “thin” auto-generated clients with bi-directional communication and minimize code reuse across service boundaries

For Type 2 decisions, choose a path, experiment and iterate
Guiding Principle: Identify your Type 1 and Type 2 decisions; Spend 80% of your time debating and aligning on Type 1 Decisions
An Evolutionary Architecture supports **guided** and incremental change as first principle among multiple dimensions

- ThoughtWorks

Choosing a **microservices** architecture with appropriate coupling allows us to evolve across multiples dimensions
### How evolvable are the Type 1 decisions

<table>
<thead>
<tr>
<th>Change Play API</th>
<th>Previous Architecture</th>
<th>Current Architecture</th>
</tr>
</thead>
<tbody>
<tr>
<td>Asynchronous?</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Polyglot services?</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Bidirectional APIs?</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Additional Data Sources?</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

#### Known Unknowns

##### Potential Type 1 decisions in the future?

<table>
<thead>
<tr>
<th>Change Play API</th>
<th>Previous Architecture</th>
<th>Current Architecture</th>
</tr>
</thead>
<tbody>
<tr>
<td>Containers?</td>
<td></td>
<td>?</td>
</tr>
<tr>
<td>Serverless?</td>
<td></td>
<td>?</td>
</tr>
</tbody>
</table>

And we fully expect that there will be Unknown Unknowns
As we evolve, how to ensure we are not breaking our original goals?

Use **Fitness Functions** to guide change
Why **Scalability** over Throughput?

- Increase in Errors due to cache warming
- New instances were added
**Guiding Principle:** Define Fitness functions to act as your guide for architectural evolution

<table>
<thead>
<tr>
<th>Previous Architecture</th>
<th>Current Architecture</th>
</tr>
</thead>
<tbody>
<tr>
<td>Multiple Identities</td>
<td>Singular Identities</td>
</tr>
<tr>
<td>Operational Coupling</td>
<td>Operational Isolation</td>
</tr>
<tr>
<td>Binary Coupling</td>
<td>No Binary Coupling</td>
</tr>
<tr>
<td>Synchronous communication</td>
<td>Asynchronous communication</td>
</tr>
<tr>
<td>Only Java</td>
<td>Beyond Java</td>
</tr>
<tr>
<td>Data Monolith</td>
<td>Explicit Data Architecture</td>
</tr>
<tr>
<td></td>
<td>Guided Fitness Functions</td>
</tr>
</tbody>
</table>
Full Cycle Developers @ Netflix by Greg Burrell

The Full Cycle Developer

- Design
- Develop
- Test
- Deploy
- Operate
- Support

Tools
If You Don’t Know Where You’re Going, It Doesn’t Matter How Fast You Get There
by Jez Humble, Nicole Forsgren
Kotlin: Write Once, Run (Actually) Everywhere
by Jake Wharton
### Building Production-Ready Applications by Michael Kehoe

<table>
<thead>
<tr>
<th></th>
<th>Topic</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Stability &amp; Reliability</td>
</tr>
<tr>
<td>2</td>
<td>Scalability &amp; Performance</td>
</tr>
<tr>
<td>3</td>
<td>Fault Tolerance and DR</td>
</tr>
<tr>
<td>4</td>
<td>Monitoring</td>
</tr>
<tr>
<td>5</td>
<td>Documentation</td>
</tr>
</tbody>
</table>
Patterns of Streaming Applications
by Monal Daxini

Patterns Summary

<table>
<thead>
<tr>
<th>FUNCTIONAL</th>
<th>NON-FUNCTIONAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>2. Script UDF Component</td>
<td>7. Stream Processing Platform</td>
</tr>
<tr>
<td>3. The Enricher</td>
<td>8. Rewind &amp; Restatement</td>
</tr>
<tr>
<td>4. The Co-process Joiner</td>
<td></td>
</tr>
<tr>
<td>5. Event-Sourced Materialized View</td>
<td></td>
</tr>
</tbody>
</table>
Human Centric Machine Learning Infrastructure @Netflix by Ville Tuulos
Atomist - A Platform Built For Delivering Modern Cloud Native Application
Q&A

• What is gRPC?
  • “gRPC is a modern, open source remote procedure call (RPC) framework that can run anywhere. It enables client and server applications to communicate transparently, and makes it easier to build connected systems.”
  • “The main usage scenarios:
    • Low latency, highly scalable, distributed systems.
    • Developing mobile clients which are communicating to a cloud server.
    • Designing a new protocol that needs to be accurate, efficient and language independent.
    • Layered design to enable extension eg. authentication, load balancing, logging and monitoring etc.”

• What language is used in developing Atomist’s Software Delivery Machine (SDM)?
  • “…SDM is in TypeScript (or JavaScript works too), and comes with a framework designed for software delivery and development automation. Write functions to make decisions or take action, with access to all the code plus the context of the push or build or issue event. All of this is open source.”
ResMed Open Role!!!

Check them out at: careers.resmed.com

- Sr. V&V Engineer
- Agile Project Manager
- Senior Manager, Technical Product Manager
- Systems Analyst
- Manager, Platform Engineer
- Software Engineer
- Associate Software Engineer
- Senior Software Development Engineer in Test
- Associate Software Development Engineer in Test
- Project Manager, Advanced Analytics

And many more…
Thank You!