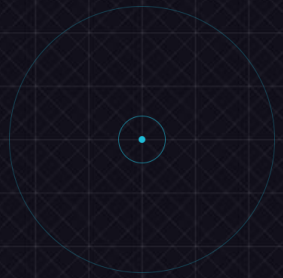


# Microservice Standardization

Susan Fowler, Uber

Software Architecture Conference San Francisco 2016

UBER

# Overview

*What We'll Cover In Today's Talk*

**Introduction**

**Terminology**

**Microservice Challenges**

**Why We Need Standardization**

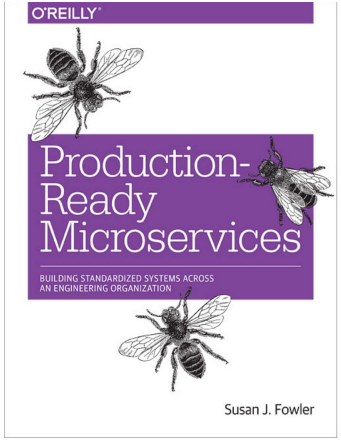
**The Goal of Standardization**

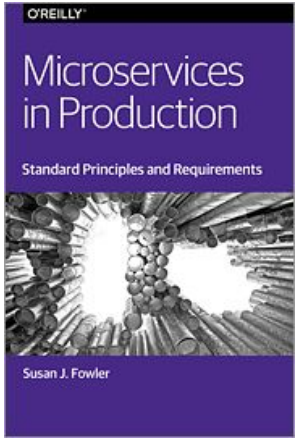
**Production-Readiness Standards**

**Implementing Standardization**

**Story Time: How Uber Standardized 1300+ Microservices**

# A little bit about myself...




ATLAS EXPERIMENT

UBER

SRE at Uber, standardizing Uber microservices

Author of *Production-Ready Microservices* and *Microservices in Production*

@susanthesquark

# Overview

*What We'll Cover In Today's Talk*

**Introduction**

**Terminology**

**Microservice Challenges**

**Why We Need Standardization**

**The Goal of Standardization**

**Production-Readiness Standards**

**Implementing Standardization**

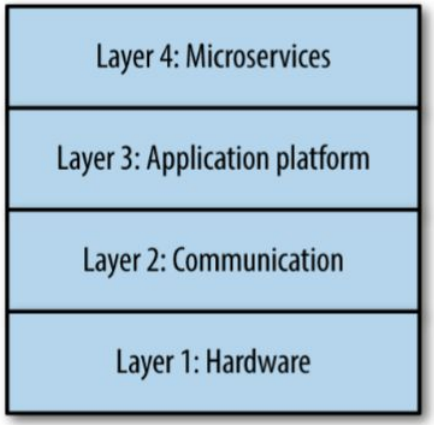
**Story Time: How Uber Standardized 1300+ Microservices**

# Terminology

***The Microservice Ecosystem***

**The overall system containing the microservices and infrastructure, which can be divided into four layers containing the microservices, the application platform, the communication layer, and the hardware layer (aka the four-layer model of the microservice ecosystem)**

# Terminology




**Layer 1: Hardware**

**servers, databases, os, resource isolation, resource abstraction, config management, host-level monitoring, host-level logging, etc**

**Layer 2: Communication**

**network, dns, rpcs, endpoints, messaging, service discovery, service registry, load balancing, etc**

**Layer 3: App Platform**

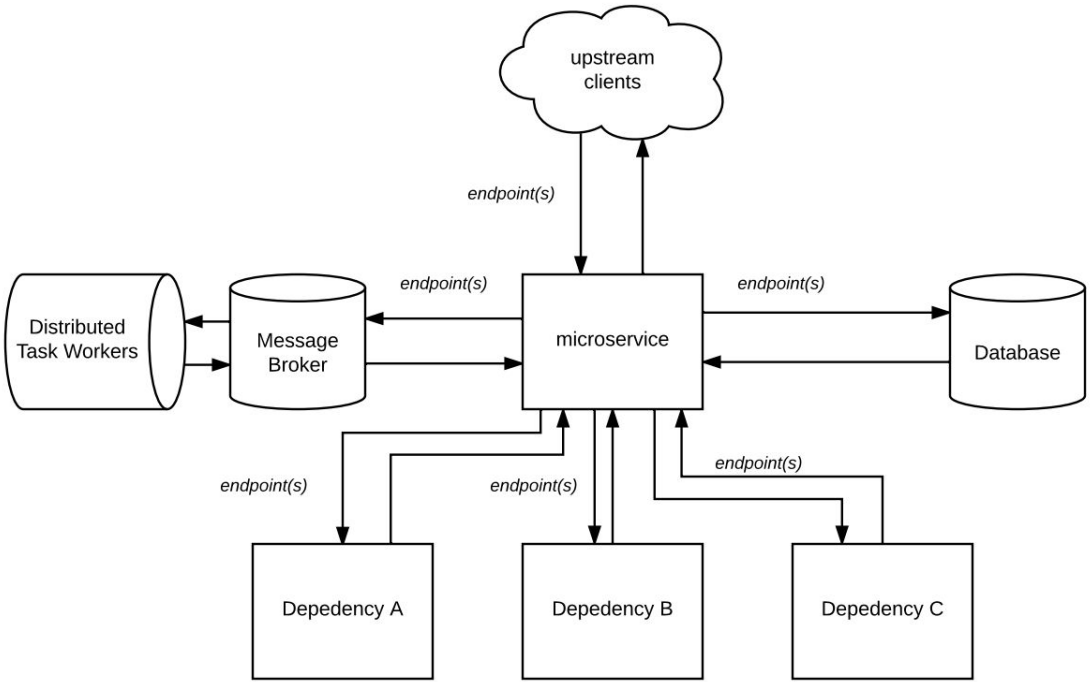
**self-service dev tools, dev environment(s), test, build, pkg, release, deployment pipeline, app-level logging, app-level monitoring, etc**

**Layer 4: Microservices**

**microservices and microservice-specific configs**

# Terminology

***Dependencies, Dependency Chains, and Clients***



# Overview

*What We'll Cover In Today's Talk*

**Introduction**

**Terminology**

**Microservice Challenges**

**Why We Need Standardization**

**The Goal of Standardization**

**Production-Readiness Standards**

**Implementing Standardization**

**Story Time: How Uber Standardized 1300+ Microservices**

# Microservice Challenges

## Challenge #1: Organizational Siloing and Sprawl

- **Inverse Conway's Law for microservices: the org structure of a company using microservices will mirror its architecture**
- **Microservice developers become like microservices (really good at doing one thing)**
- **Communication problems**
- **Operational tasks must be shouldered by development teams**

# Microservice Challenges

## Challenge #2: More Ways to Fail

- **Microservices are parts of large and complex distributed systems**
- **The more distributed the system, the more ways it can (and will) fail**
- **Each microservice becomes a point of failure**

# Microservice Challenges

## Challenge #3: Competition for Resources

- **Microservice ecosystem is just like any other ecosystem**
- **Hardware resources are scarce**
- **Engineering resources are scarce**
- **Difficult to prioritize**
- **Difficult to scale**

# Microservice Challenges

## Challenge #4: Misconceptions about Microservices

- **Myth: Microservices are the Wild West**
- **Myth: Free reign over architecture decisions**
- **Myth: Freedom to choose any programming language**
- **Myth: Freedom to choose any database**
- **Developers think “microservices” = “build a service that does one thing extraordinarily well; do whatever you need to do, and build it however you want - just make sure it gets the job done”**

# Microservice Challenges

## Challenge #5: Technical Sprawl, Technical Debt

- **Everyone uses their favorite tools**
- **Everyone deploys with custom scripts**
- **Everyone builds custom infrastructure**
- **A thousand ways to do each thing**

# Microservice Challenges

## Challenge #6: Inherent Lack of Trust

- **Microservices live in complex dependency chains, completely reliant on each other**
- **No way to know for sure that dependencies are reliable**
- **No way to know that clients won't compromise your microservice**
- **No trust at organizational, cross-team, or team levels**
- **No way of knowing that microservices can be trusted with production traffic (that is, no way of knowing if microservices are production-ready)**

# Overview

*What We'll Cover In Today's Talk*

**Introduction**

**Terminology**

**Microservice Challenges**

**Why We Need Standardization**

**The Goal of Standardization**

**Production-Readiness Standards**

**Implementing Standardization**

**Story Time: How Uber Standardized 1300+ Microservices**

# Why We Need Standardization

***Reality: microservices are not isolated systems***

**They are part of the microservice ecosystem,**

**and belong in complex dependency chains**

**No microservice or set of microservices should compromise the integrity of the overall product or system**

# Why We Need Standardization

***Solution:***

**Hold all microservices to high architectural, operational, and organizational standards**

**A microservice that meets these standards is deemed “production ready”, meaning it can be trusted with production traffic**

# Why We Need Standardization

## Approach #1: Local Standardization

- **Determine standards on a microservice-by-microservice basis**
- **Figure out what requirements are appropriate for each individual service, and go from there**

## Problems:

- **Doesn’t establish organizational, cross-team, team trust**
- **Adds to technical sprawl and technical debt**
- **Not scalable**
- **Don’t know if services are *production-ready***

# Why We Need Standardization

## Approach #2: Global Standardization

- **Determine standards that apply to all microservices within the ecosystem**
- **Make them general enough to apply to every microservice**
- **Make them specific enough to be quantifiable and produce measurable results**

## Problems:

- **Hard to determine from scratch what appropriate standards are**
- **Hard to figure out standards that apply to all microservices and actually make a difference**

# Overview

*What We'll Cover In Today's Talk*

**Introduction**

**Terminology**

**Microservice Challenges**

**Why We Need Standardization**

**The Goal of Standardization**

**Production-Readiness Standards**

**Implementing Standardization**

**Story Time: How Uber Standardized 1300+ Microservices**

# The Goal of Standardization

**Let’s switch gears for just a moment, and talk about availability**

**Availability = uptime/(uptime+downtime)**

**Availability is an awesome way to quantify trust - can use SLAs for availability between microservices as a way to measure trust**

# The Goal of Standardization

**However…**

- **Availability isn’t itself a microservice standard**
- ***It gives no guidance as to how to architect, build, or run a microservice***
- **Telling developers to make their service more available is not useful, and will get you nowhere.**
- **We can use availability as a goal, and ask: “what makes a microservice more available?”**

# The Goal of Standardization

## Approach #1: Local Standardization

- **Determine standards on a microservice-by-microservice basis**
- **Figure out what requirements are appropriate for each individual service, and go from there**

## Problems:

- **Doesn’t establish organizational, cross-team, team trust**
- **Adds to technical sprawl and technical debt**
- **Not scalable**

# Overview

*What We'll Cover In Today's Talk*

**Introduction**

**Terminology**

**Microservice Challenges**

**Why We Need Standardization**

**The Goal of Standardization**

**Production-Readiness Standards**

**Implementing Standardization**

**Story Time: How Uber Standardized 1300+ Microservices**

# Production-Readiness Standards

## KEY PRINCIPLES

**stability**
**reliability**
**scalability**
**performance**
**fault-tolerance**
**catastrophe-preparedness**
**monitoring**
**documentation**

***Each standard is accompanied by quantifiable requirements that produce measureable results***

# Production-Readiness Standards

***Stability***

**We get increased developer velocity with microservices, so there are more changes, more deployments, and more instability**

**Stability allows us to reach availability by giving us ways to responsibly handle changes to microservices**

# Production-Readiness Standards

***Reliability***

**A reliable microservice is one that can be trusted by its clients, dependencies, and the ecosystem as a whole**

**Stability and reliability are inextricably linked, and most stability requirements have accompanying reliability requirements**

# Production-Readiness Standards

## Stability and Reliability Requirements

- **Standardized development cycle**
- **Code is thoroughly tested through lint, unit, integration, and end-to-end testing**
- **Test, packaging, build, and release process is completely automated**
- **Standardized deployment pipeline, containing staging, canary, and production phases**
- **Dependencies are known, and there are backups, alternatives, and/or caching in place in case of failures**
- **Stable and reliable routing and discovery**

STABILITY AND RELIABILITY

# Production-Readiness Standards

***Scalability***

**Microservices need to scale appropriately with increases in traffic**

**Scalability is essential for availability - a microservice that can't scale with expected growth has increased latency, poor availability, and (in most cases) a drastic increase in # of incidents and outages**

# Production-Readiness Standards

***Performance***

**Scalability and performance are linked: scalability deals with how many requests a microservice can handle, and performance is related to how well the service can process those tasks**

**A performant microservice handles requests quickly, processes tasks efficiently, and properly utilizes resources**

# Production-Readiness Standards

## Scalability and Performance Requirements

- **Qualitative and quantitative growth scales**
- **Efficient use of hardware resources**
- **Resource bottlenecks and requirements have been identified**
- **Automated, planned, and scheduled capacity planning**
- **Dependencies scale with each service**
- **Each service scales with its clients**
- **Well-known traffic patterns**
- **Traffic can be re-routed**
- **Written in a scalable programming language**
- **Handles and processes tasks in a performant way**
- **Handles and stores data in a scalable and performant manner**

SCALABILITY AND PERFORMANCE

# Production-Readiness Standards

***Fault-Tolerance and Catastrophe-Preparedness***

**Microservices live in complicated, messy ecosystems in complex dependency chains and can (and do) fail all of the time and in every way imaginable**

**To ensure availability, microservices need to be able to withstand internal and external failures**

# Production-Readiness Standards

## Fault-Tolerance and Catastrophe-Preparedness Requirements

- **No single point of failure**
- **All failures scenarios and possible catastrophes have been identified**
- **Tested for resiliency through code testing, load testing, and chaos testing**
- **Automated failure detection and remediation**
- **Standardized incident and outage procedures in place**

FAULT-TOLERANCE

# Production-Readiness Standards

***Monitoring***

**Proper monitoring allows us to know the state of the system at all times**

**Second most common cause of outages is lack of good monitoring: if you're not aware of the state of the system, you won't know when the system fails**

# Production-Readiness Standards

## Monitoring Requirements

- **All key metrics are identified and monitored at the host, infrastructure, and microservice levels**
- **Logging accurately reflects the past state of the microservice**
- **Dashboards are easy to interpret and display all key metrics**
- **All alerts are actionable and defined by signal-providing thresholds**
- **Dedicated on-call rotation responsible for monitoring and responding to incidents and outages**
- **Standardized on-call procedure for handling incidents and outages**

MONITORING

# Production-Readiness Standards

***Documentation and Understanding***

**Documentation shaves off a lot of the technical debt, as does understanding of the service at the organization, team, and developer levels**

# Production-Readiness Standards

## Documentation and Understanding Requirements

- **Microservice has comprehensive documentation**
- **Documentation is regularly updated**
- **Documentation contains all essential information about the microservice**
- **It is well understood at the developer, team, and organizational levels**
- **It is held to a set of production-readiness standards and corresponding requirements**
- **Its architecture is reviewed and audited frequently**

DOCUMENTATION

# Overview

*What We'll Cover In Today's Talk*

**Introduction**

**Terminology**

**Microservice Challenges**

**Why We Need Standardization**

**The Goal of Standardization**

**Production-Readiness**

**Production-Readiness Standards**

**Implementing Standardization**

**Story Time: How Uber Standardized 1300+ Microservices**

# Implementing Standardization

*Now that we have a set of production-readiness standards, what's next?*

## Step One: Get buy in from all levels of the organization

Standardization needs to be adopted and driven from the top-down and from the bottom-up.

## Step Two: Determine your organization's production-readiness requirements

Production-readiness requirements need organizational context in order to be effective

## Step Three: Make production-readiness part of the engineering culture

Standardization needs to be seen not as a hindrance or a gate to development and deployment, but a guide for building and running the best possible microservices

# Overview

*What We'll Cover In Today's Talk*

**Introduction**

**Terminology**

**Microservice Challenges**

**Why We Need Standardization**

**The Goal of Standardization**

**Production-Readiness Standards**

**Implementing Standardization**

**Story Time: How Uber Standardized 1300+ Microservices**

# Standardizing 1300+ Microservices at Uber

UBER

# Questions?

@susanthesquark