How to Build, Test and Deploy Large Distributed Systems for Real Time Media Processing
Who we are

Sebastian Dähne

Started at VidSoft as intern - 2001

Mostly SysAdmin

Full Time at Citrix / GetGo / LogMeIn - 2011

C++ Backend Programming

started with Tooling for Deployments - 2014

Manager for DevOps and Security - 2016

Robert Lübke

2011-2015: Researcher in Computer Networks Group, TUD

PhD about network emulation in large-scale testing

2015 - now: Citrix / GetGo / LogMeIn (Senior Software Development Engineer in Test)

– Test Automation
– Network Testing (local, lab, data center, AWS)
– Load / Stress / Performance Testing
– Distributed Testing of (JavaScript) applications
What's it all about?

- in the business of building large distributed systems for real time media for 19 years
- running a large scale SaaS business for online conferencing
- special challenge: how do we deploy, manage and test such systems?
- very fast developing area, lots of new concepts
- today: overview of where we currently are in this journey
Structure

Introduction LogMeIn

Build – Test – Deploy – Operate

Cloud-Native

Summary

Working at LogMeIn
WE UNLOCK THE POTENTIAL OF THE MODERN WORKFORCE.
Communications & Collaboration

“make it easy to connect quickly and professionally, whether across town or across the globe”
Product Example: GoToMeeting

**HD Video Conferencing**
To really communicate, you need to see people’s faces.

**Share Your Screen**
Present and collaborate right from your iPad or computer.

**Audio Included**
VoIP or phone? Everyone gets their own choice. And toll-free is also available.

**Meeting Recording**
Save conversations and distribute them to those who couldn’t attend.
Serving the Global Customer

- billions of voice minutes / year
- millions of users per day
- hundred thousands of concurrent users in real-time sessions

→ Globally distributed servers, 24/7 service availability (99.9% target)
→ Problems: geo/locality, scale, reliability, cost efficiency
RTC Group: Reliable Core Services

Teams:

- Jersey City
- Santa Barbara
- Karlsruhe
- Dresden (60+ Engineers)
  - Video Server Platform
  - Audio & Video Endpoints
  - Video Coding
  - Audio Signal Processing
  - Data Analysis / Machine Learning
  - Infrastructure/Tools Team
  - DevOps
Technology Areas

- Video
- Audio
- Processing
- Mixing
- Protocols
- Distribution
- Bridges
- Capture
- Conferences
- Echo
- Voice Quality
- Storage
- Rendering
- AWS
- Technologies
- Server
- Coding
- Large-scale
- Activity
- Control
- Voice
- Suppression
- PSTN
- Signaling
- Layout
- Participants
- Signal
- VP
- Automation
- Detection
- Platform
- Noise
- Gain
- VoIP
- Test
- Automatic
- Transport
- Docker
- Cloud
- Public
- Voice cancellation
- Control
- Suppression
- Signaling
Changing Paradigms

- Agile
- Testing
- DevOps
- Cloud-native
Building Software

*How are we building?*

Compile -> Package -> Store

- Compiling in a Docker container
- Packaging Docker Images
- Storing in a Docker Registry
Docker

Why do we want to use Docker?

• Build: define our own build environment, use same infrastructure
• Dev/Test: use the same Artifact in the entire chain
• Deploy: abstract from the operating system
  • Potentially make use of container orchestration
Building in Jenkins (automation server)

*Is Jenkins the best tool to build C++?*

- **Jenkinsfile**: Build System as Code
- **Jenkins** – good workflow engine
- **Works very good** with abstracting the build environment into the docker container
“But testing is not about quality. Quality has to be built in, not bolted on, and as such, quality is a developer task.”

(How Google Tests Software)
Everyone Owns Product Quality

No test department as gatekeeper

No quality after the fact

Testers / Quality Engineer are part of development team

Whole team builds test automation at various stages
Paradigm Change

How do we ensure software quality throughout this process?

Image source: https://leanpub.com/AgileTesting/read
Testing Today is Automated

Manual test execution
- no longer fast enough to keep pace with short release cycles
- pretty complex, requires skilled testers
- no scaling by hiring testers

Solution
- Test Automation on all stages of the development process
- Separate teams working on test automation tools in advance
Testing on the different layers

Test cone for automation
Shows types of tests and how much of each type we would like to have in an ideal world.
Who builds all the automated tests?

Different concepts

• dedicated QA people in the teams focusing on test automation.
• unit tests are written by developers, ideally as part of test driven development (TDD)
• Testing as whole team approach: test folks understand and write production code, especially also fixes, and developers write more complex tests
• dedicated people in the QA team building larger test infrastructure and frameworks
Bring your own test – we take care of the rest

- Developers writing integration, component, and systems tests in TypeScript using byot.js
- Focus on distributed testing for JS/Native components on the lower levels.
- Takes care of resource provisioning, deployment, orchestration, artifact collection, debugging, ...
Examples - 2

Audio Quality Assessment

• REST service to automatically evaluate audio quality, similar to a subjective Mean Opinion Score (MOS)

BlackCAT

• Inject and record audio files in multi-platform and multi-OS systems

OpenWRT Degrader

• Artificially worsen connections that go over a configured router
Deployment Process

*Divided into 2 parts*

Creating the infrastructure necessary

• We call that provisioning

• Using Amazon Web Services (AWS)

Pushing the service onto the infrastructure

• We call that deployment

• Using Docker
From state: on-premise

Where did we start?

- Bare metal servers or Xen VM’s
- RHEL5/6
- Installation via copy & rpm install
- New installation took days (hardware, network, os, application)
Creating the Infrastructure - Infrastructure as Code

Every part of the infrastructure is based on code

- Checked in with history
- Can be automatically tested
- Can be reviewed
Creating the Infrastructure - Infrastructure as Code

**Tools/Services**

- (Amazon Web Services) EC2 (Elastic Compute Cloud)
- Cloudformation (JSON DSL for creating AWS Resources)
- Sparkleformation - abstract Cloudformation in a programmatic way
- Route 53 (AWS DNS Solution)
  - Geo-based “Routing”
How does it work - Provisioning

- Sparkleformation
- Cloudformation templates: Compute resources, Load balancers, DNS
- Amazon Web Services
- GIT
- Jenkins
How does it work - Deploy

- **GIT**
  - Image config
  - Service config

- **Jenkins**

- **Fleet Configuration Service**
  - Datacenter
  - Cluster
  - Node

- **Docker Registry**

- **AWS EC2 Host**
Is that it?

*Automated but static*

- Programmatic creation of infrastructure
- Programmatic deploy of services
- But somewhat static, 2 interfaces
- > need some more to scale dynamically
Autoscaling

*Instances are self sufficient*

- Use of AWS Lambda to enable self-config
- Use of AWS Autoscaling groups to enable scaling
How does it work - Autoscaling

**GIT**
- Image config
- Service config

**AWS**
- Route53
- Elastic IP

**Fleet Configuration Service**
- Datacenter
- Cluster
- Node

**Jenkins**

**Cloudformation templates**
- AutoScaling Group
- Load balancers

**AWS Lambda**

**AWS AutoScaling Group**
- AWS EC2 Host
  - Cloud Init
Build
Test
Deploy
Operate
Pets vs. Cattle

**Pets**

- Servers of pair of servers treated as indispensable
- Manually Built and managed, can never be down
- E.g. Mailserver.somewhere.net, webserver.somewhere.net

**Cattle**

- More than 2 servers – built automatically, looking alike
- Designed for failure - failed servers will be recreated
- E.g. server[01..20].somewhere.net
Operational Challenges: LogFiles

Where do we store and analyze our logs?

- Services produce log files (lots of them)
- Hard to analyze distributed

-> Central logging service
  - SPLUNK
  - Elastic Search

- Hint: Please be reasonable with amount
Operational Challenges: Monitoring

*Push vs. Pull, TICK and Wavefront*

- Cloud is dynamic in nature
- Central collector needs extra registration
- Rather push then pull
- Telegraf, Influx, Chronograf (Grafana), Kapacitor
Monitoring Strategy

- **InfluxDB**
  - Dev defined
  - Low retention time
  - Low SLA
  - Cheap

- **Grafana**
  - Dev defined
  - High retention time
  - High SLA
  - More expensive

- **Wavefront**
  - DevOps defined
  - High retention time
  - High SLA
  - More expensive

- **Wavefront Proxy**

- **Service Host/Env**
  - Service
  - CPU/MEM
And now?
Paradigm Change - Transition to a Cloud-native Environment

*Our way to the Cloud*

- Started with VM’s On Premise -> Cloud
- Introduced containers as abstraction layer
- Infrastructure as code and integrated in all steps (build, test, deploy)

Are we done?
Paradigm Change - Transition to a Cloud-native Environment

The only constant is Change

- Are we done? – No!
- Design for failure and non-permanent resources
- Design to update quickly
- New technologies will come up and help

But Why – it’s working, isn’t it?
e.g. Meltdown / Spectre

*When sh** hits the fan*

- **What**
  - Design problems in recent CPU architectures
  - Industry wide problem
  - Need to react fast once solutions are available

- **Meltdown**: Require patching/rebooting
- **Spectre**: recompile (retpoline) and re-deployment
Ever changing system / Future

Stay on top of technology

• Architecture and technology keeps evolving
• Container orchestration tools
• E.g. Amazon Container Service, Kubernetes
• Serverless (Function as a Service)
• E.g. Amazon Lambda
Software Engineering is more than “only” developing

We see the future developer as an engineer working in a production team supported by complex processes and tools.

The How of software development (Software development life cycle) will change continuously at a fast pace, so you should be prepared to think about it a lot, as this needs to be adopted and driven by the development teams.

Software development is more than coding – the developer is only done, once the customer has the (working) software in his hands. So it’s also testing, deployment and maintenance and it all has to be considered as one single process, which is owned by the development teams.
Working at LogMeIn
What we expect new people to know

Familiar with concepts for unit testing (i.e. jMock)

Familiar with common frameworks for integration testing (i.e. webdriver/selenium)

Familiarity with continuous integration, continuous deployment and the respective tools (i.e. bamboo, jenkins)

Know your theory about architecture of distributed systems, communication paradigms, network protocols

Obviously, practical programming experience (we still work mainly with C++, with some JavaScript, Java, Ruby and Python thrown in)

Learn and do some stuff outside of your curriculum
Possibilities for Students at LogMeIn

During the studies
• Working student (~12€/h + bonus based on performance)
• Internships (mandatory, also possible in other LogMeIn offices), for example 8 weeks Germany + 12 weeks USA
• Komplexpratikum with Computer Networks Group
• Student theses (Master, Bachelor, Diploma, Großer Beleg) – some announced topics, but open to proposals

Post-Graduation: https://www.logmeininc.com/careers + unsolicited applications
• Google
  – How Google Tests Software - Whittaker, James A.; Arbon, Jason; Carollo, Jeff (2012-03-23).
  – Past Test Automation Conferences: https://developers.google.com/gtac
• Gene Kim: The Phoenix Project (2013)
Thank you.