Bazooka — Continuous Deployment at SoundCloud

Alexander Simmerl
Matt Proud

http://goo.gl/VNxpa

Talk Format

1. What is Bazooka?
2. Why was it made?
3. How does it work?
4. What were the results and reflections?
5. Where do we go from here?

What is Bazooka?

deployment for 12factor.net apps

- git based
- process scaling
- process lifecycle management
- process visibility

Social Challenges: Growth

Company started with a team that fit in one room:

- Everybody knew everything or could just ask the adjacent person.
- Little incentive to document, making wisdom tribal.

Historical deployment processes were manual and cumbersome but understood.

Social Challenges: Growth

Engineering outgrew its office and spans three buildings:

- Little time available to document launch or reiteration processes, effectively siloing wisdom.
- New components have disparate launch and deployment processes.
- Time consuming to onboard new-hires and train them in deploying to production their first pull requests.

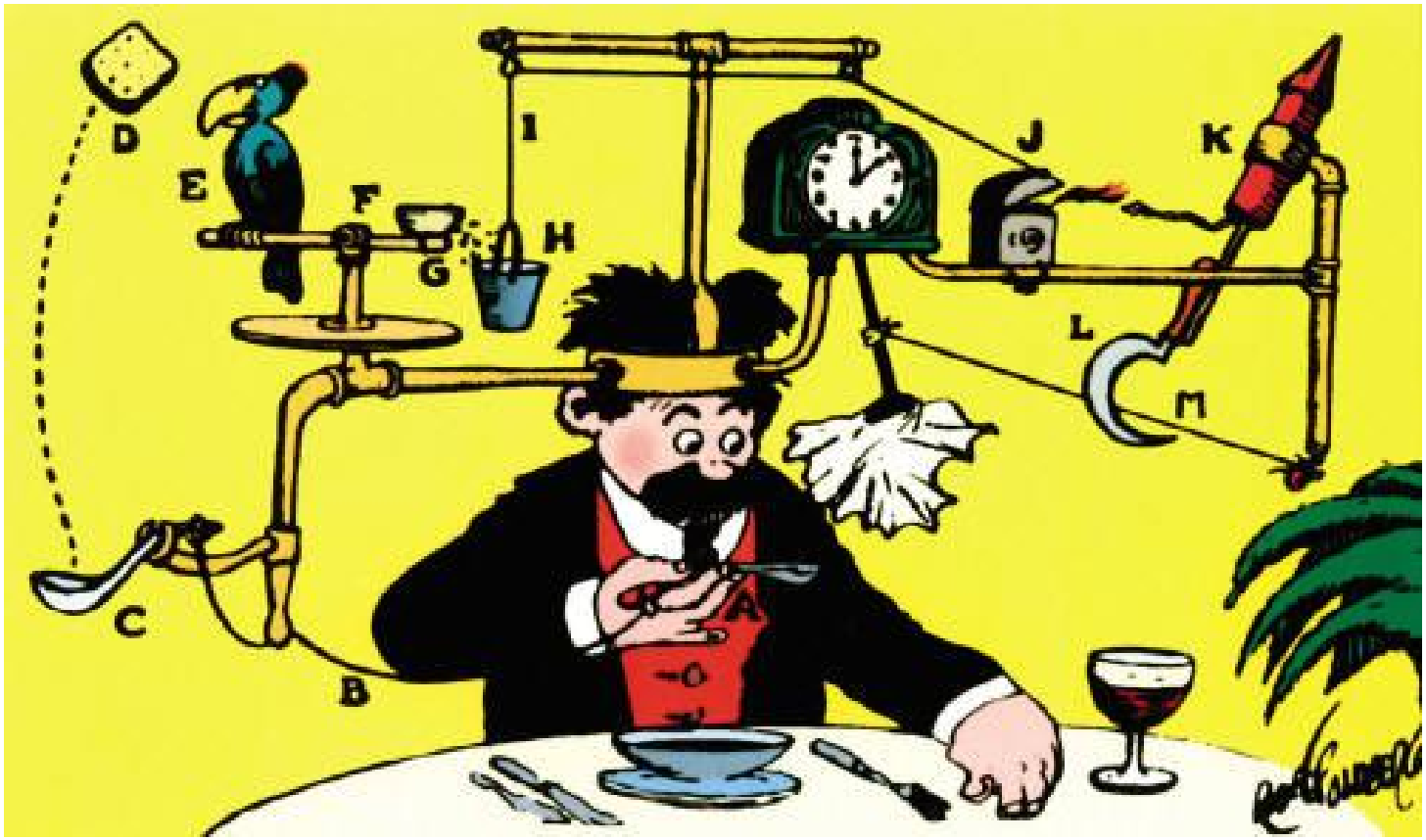
Social Challenges: Complexity

…, which led to an oversubscribed systems engineering team responsible for machine resources and as primary escalation point:

- Development teams could no longer care for themselves.
- Lack of common processes and documentation and scaffolding raised high coefficient of friction.

Sentiment: “We feel slow now and demoralized. How do we get our mojo back?”

Rube Goldberg Complexity ≠ Agility



Technical Challenges: Velocity

Main site was powered by a single monolithic codebase:

- Hard for teams and services to advance independently.
- No understanding of interaction between services through well-defined APIs.
- Only a handful of independent services outside of the monorail.

Technical Challenges: Velocity

The important deploys were slow:

- Scripts deployed to a static list of tens of hosts.
- Same resource heavy steps were happening on all hosts.
- If one host failed, the whole deploy failed.
- Not isolated from external dependencies:
  - github
  - rubygems
  - etc.

Technical Challenges: Capacity

Statically assigned instance pools for current services:

- Only a subset of instances can be restarted in parallel to avoid backlog.
- Rollbacks are slow as the same deploy mechanisms were used.
- Additional allocation of capacity needs manual interaction.
- Utilization per host inefficient.

Technical Challenges: Capacity

Scaling steps for new services:

- During prototype/low-usage phase placed on small VMs.
- Repeated questions when the VM is not enough:
  - What HA requirements?
  - What server class?
  - How many instances?
  - How and what to monitor?
  - Who can provision?

Technical Challenges: Cohesion

Many ways to deploy:

- Each new project independently decided what method to use.
- Variety of solutions:
  - capistrano
  - chef
  - bash
  - git remote
- No shared patterns.

Goals

Distill deployment into one common interface to

- ease exposing a new service to the public
- lower experimental iteration costs
- encourage building of small projects
- offer production job isolation
- encourage common process development
- provide predictable human interfaces

Non-Goals

- Automatic scaling for load
- Support for stateful jobs
- Inherent workflows for batch jobs

Workflow: init

```
$ cd src/goto-rocket
$ bazooka init
app 'goto-rocket' registered
[bazooka-rm] remote repository created
[bazooka-rm] git config updated
[bazooka-rm] hooks setup
remote entry created
```

Workflow: build

```
$ git push bazooka master
remote: change-set accepted
...
remote: uploading container to file storage
remote: registering revisions in coordinator
...
remote: pty registered: web
remote: build finished
```

Workflow: procs

```
$ bazooka procs
APP            PROC   PORT   MEMLIMIT   INSTANCES
goto-rocket    fail   8509   -          0
goto-rocket    job    8510   -          0
goto-rocket    web    8511   -          0
```

Workflow: scale

```
$ bazooka scale -n 1 web
web@16f7cbf  0 -> 1  .
```

Workflow: ps

```
$ bazooka ps
ID       APP           PROC  REV      IP           PORT
4617216  goto-rocket   web   16f7cbf  1.2.3.4      21880
```

Technical Overview: Terminology

app: versioned codebase with n services

proc: named command with n instances

rev: point-in-time snapshot of the app

service: tuple of app & proc

instance: running proc

Technical Overview: Components

- **bazooka**: command-line interface
- **bazooka-pm**: process manager
- **bazooka-rm**: repository manager
- **bazooka-proxy**: proxy config manager
- **doozerd**: point of integration for the above
- **visor**: coordinator client, used by the above

Technical Overview: Statistics

- Overall
  - # Apps
  - # Proctypes
  - # Revisions
  - # Instances
- Daily
  - # Apps
  - # Proctypes
  - # Revisions
  - # Instances
- **13** application hosts, **2** proxies, **1** builder
- Serving ~**5** KQPS

Reflections: Outcomes

It …

- takes only an hour or two for a new user to push a service for the first time and avail it to the outside!
  - Down from potentially several days.
- takes minutes for subsequent iterations.

Small interaction surface promotes principle of least surprise.

Reflections: Patterns to Design By

I/O

- Assume read-only local filesystem.
- Avoid swapping.
- Logging to STDOUT with known schema.

Workflow and Lifecycle

- No manual intervention in starting or stopping a job.
- Idempotency.

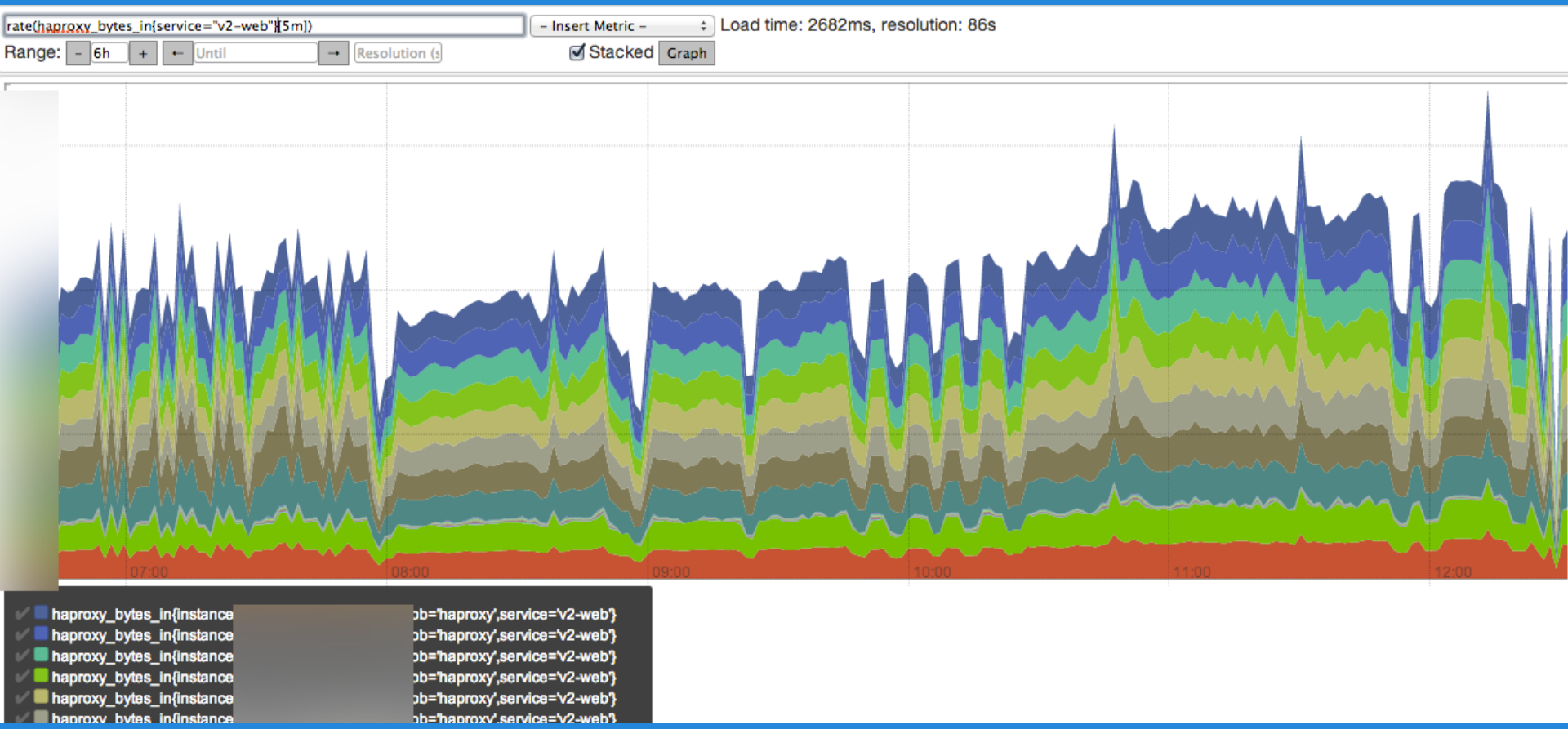
Reflections: Patterns to Design By

Data-Driven

- Good release qualification process.
- Granular telemetry to understand your stack and differentiate behaviors across subjects.

Backend Dependencies

- Graceful degradation mode (e.g., circuit breaker)
- Know locality of dependencies
- Interrelease interface compatibility



Aggregate Service Telemetry: Per-Bazooka Instance Throughput for "v2-web"

Reflections: Patterns to Design By

Understand your traffic layer

- Cache-control
- Termination
- Routing

Reflections: What to do Again?

Keep the scope small, simplicity first:

- Use a methodology with hard constraints (12factor)
- Don't make quick tradeoffs, educate instead.
- Decouple components (buildtime, runtime, routing, discovery)
- Build for global consistency.
- Build a system that converges.

Reflections: What to do Again?

Have small building blocks to iterate quickly:

- Center implementation around a lib, which encodes the domain.
- Use lib to build small, quick to evolve binaries.
- Provide tools that are easy scriptable (cli).
- Use versioned schemas to avoid data corruption.
- Use Go again for similar infrastructure tasks.

Reflections: What to do Differently?

Failure as the default case:

- Build for simple and fast recovery.
- Move from host-local to distributed instance supervision
- Invest in tooling for operations and testing (config, test clusters, etc.).
- Use and define contracts over assumptions, to avoid later maintenance overhead.

Reflections: What to do Differently?

Build a healthy community:

- Engage users early and often.
- Keep the documentation detailed and up-to-date.
- Engage users early and often!

Reflections: Technical Challenges

- SPOFs in underlying SoundCloud infrastructure.
- Doozer: Maintainership and failure modes.
- Further build pipeline improvements.
- Capacity planning is informal.
- Lack of service discoverability and addressing.
- Have better resource isolation.

Roadmap: Telemetry

Becoming data-driven ourselves using newly-built, in-house timeseries collection and computation system called Prometheus to

- provide per-application monitoring
- enable proactive capacity planning

We already have aggregate system telemetry, though!

Roadmap: Service Discovery

Ephemeral port allocation should not make life difficult. We're building service discovery to …

- connect with hosted applications with monitoring infrastructure, and
- facilitate easy interservice integrations.

Roadmap: Multihoming

Bazooka is designed in a manner to support local serving zones.

That said, …

The tooling around it could be polished to make this workflow more seamless.

Contributing / We're Hiring

- Open sourcing forthcoming
- Follow: backstage.soundcloud.com
- Visit our meet up: lanyrd.com/cpbkq.
- We're hiring: goo.gl/FVIfH.

Questions and Contact

- {alx,matt}@soundcloud.com
- These slides: goo.gl/VNxpa.