Internal Services Have Customers Too!

KC Braunschweig
Production Engineer
SCaLE 17x – March 2019
Internal Services Have Customers Too!
Internal Services Have Customers Too!
Internal Services Have Customers Too!
Who Am I?

- SCaLE Volunteer

@kcbraunschweig
Who Am I?

- SCaLE Volunteer
- Ticketmaster - Web Operations
- Edmunds.com - Systems Engineering
Who Am I?

• SCaLE Volunteer
• Ticketmaster - Web Operations
• Edmunds.com - Systems Engineering
• Facebook - Production Engineering
  • OS & Config Management (Chef)
  • Logging Infrastructure (Scribe, Hadoop & LogDevice)
  • Coordination Infrastructure (Apache Zookeeper)
Agenda

• Intro
• Facebook Service Examples
• Service Maturity Scenarios
• Conclusions
Facebook Service Examples
Facebook Examples
Scribe

• Originally a purpose built logging framework for dozens of use cases
  • “Today we have well over 100 applications using this” – Bobby Johnson 2009 [2]

• Now the transport layer for all logging, stream processing
  • Many 1000s of categories and >1TB/s [3]

• 10+ years becoming a massive multi-tenant service

Facebook Examples

Chef

• All systems-level configuration at Facebook
  • Designed for a *small team* to manage a *massive* fleet
  • Delegate responsibility to customer teams
• “Have 4 people manage 10s of thousands of heterogeneous systems” – Phil Dibowitz 2014 [1]
• ~6 years of maturing

[1] “Really large scale systems configuration” https://www.youtube.com/watch?v=rEWHmk8vBYk
**Facebook Examples**

Zookeeper

- Originally backing 2 major use cases:
  - Service discovery system
  - Application configuration distribution system
  - One team with a handful of ensembles
- Now Zookeeper as a Service
  - Hundreds of ensembles [4]
  - One Zookeeper team with many customer teams

Service Maturity Scenarios
Facebook Examples

Plan of attack

• You don’t get to pick where to start
• What’s right? What’s wrong?
• What do we need to make things better?
Config Smell

Monitoring Zookeeper ensembles
Config Smell
Monitoring Zookeeper ensembles

• ”Where are the monitoring configs?” – Zookeeper team n00b

* Ensembles have names e.g. zk.global.42
Config Smell
Monitoring Zookeeper ensembles

• ”Where are the monitoring configs?” – Zookeeper team n00b

```
filter(zk\.global\.(0[389]|29|4[2-8]|6[589]|72|103))
```

* Ensembles have names e.g. zk.global.42
Config Smell
Monitoring Zookeeper ensembles

• ”Where are the monitoring configs?” – Zookeeper team n00b

filter(zk\.global\.(0[389]|29|4[2-8]|6[589]|72|103))
# and 2 other places with slight variations

* Ensembles have names e.g. zk\.global\.42
Config Smell
Monitoring Zookeeper ensembles

• ”Where are the monitoring configs?” – Zookeeper team n00b

filter(zk\.global\.(0[389]|29|4[2-8]|6[589]|72|103))


# and 2 other places with slight variations

* Ensembles have names e.g. zk\.global\.42
Config Smell

Growing number of ensembles

Wormhole [5] team uses a growing subset of ensembles

Wormhole monitoring is slightly different due to their workload

Config Smell
Monitoring Zookeeper ensembles

• That’s a little better

```bash
# Special wormhole ensembles - keep updated!
WH=filter(zk\global\.(0\[389]|29|4\[2-8]|6\[589]|72|103))
```
That’s a lot better

```python
filter(get_ensembles_by_customer('wormhole'))
```
Config Smell
Monitoring Zookeeper ensembles

- That’s a lot better or is it?

```python
filter(get_ensembles_by_customer('wormhole'))
filter(get_ensembles_by_customer('wormhole2'))
```
Config Smell
Monitoring Zookeeper ensembles

• That’s a lot better or is it?

```python
filter(get_ensembles_by_customer('wormhole'))
filter(get_ensembles_by_customer('wormhole2'))
filter(get_ensembles_by_customer('stargate'))
filter(get_ensembles_by_customer('lorem'))
filter(get_ensembles_by_customer('ipsum'))
filter(get_ensembles_by_customer('adnauseum'))
...```
Config Smell
Monitoring Zookeeper ensembles

• How about this

```python
filter(get_ensembles_by_sla('hipri'))
```
Config Smell
Monitoring Zookeeper ensembles

• Or better yet

```python
for sla, ensembles in get_ensembles_by_sla().items():
    # do stuff for each sla
    filter(ensembles)
```
What do we need?

- Separate customer metadata from service implementation
- Define scalable service offerings
- Canonical store of customer metadata
Customer Metadata
Scribe Categories

Background

- Log events are written to scribe categories
- Categories must be registered
- Registration has required fields
$ dmv find kctest1 -f json

[{
   "Category": "kctest1",
   "Blacklist Threshold": "1GB",
   "Encryption": "Yes",
   "Importance": "normal",
   "MailTo": ",",
   "MaxRate": "1MB",
   "Modified": "2018-10-08T09:00:17",
   "Modified By": "security_oncall",
   "Oncall": "scribe_oncall",
   "Owner": "kcb",
   "Retention": 1,
}]

Customer Metadata
Scribe Categories
$ dmv find kctest1 -f json

[
{
    "Category": "kctest1",
    "Blacklist Threshold": "1GB",
    "Encryption": "Yes",
    "Importance": "normal",
    "MailTo": "",
    "MaxRate": "1MB",
    "Modified": "2018-10-08T09:00:17",
    "Modified By": "security_oncall",
    "Oncall": "scribe_oncall",
    "Owner": "kcb",
    "Retention": 1,
}]

Customer Metadata
Scribe Categories
Customer Metadata

Scribe Categories

• + Know who our customers are
Customer Metadata
Scribe Categories

```bash
$ dmv find kctest1 -f json
{
  "Category": "kctest1",
  "Blacklist Threshold": "1GB",
  "Encryption": "Yes",
  "Importance": "normal",
  "MailTo": "",
  "MaxRate": "1MB",
  "Modified": "2018-10-08T09:00:17",
  "Modified By": "security_oncall",
  "Oncall": "scribe_oncall",
  "Owner": "kcb",
  "Retention": 1,
}
```
Customer Metadata

Scribe Categories

- + Know who our customers are
- - Implementation leakage
$ dmv find kctest1 -f json
[
{
    "Category": "kctest1",
    "Blacklist Threshold": "1GB",
    "Encryption": "Yes",
    "Importance": "normal",
    "MailTo": "",
    "MaxRate": "1MB",
    "Modified": "2018-10-08T09:00:17",
    "Modified By": "security_oncall",
    "Oncall": "scribe_oncall",
    "Owner": "kcb",
    "Retention": 1,
}]

Customer Metadata

Scribe Categories

• + Know who our customers are
• - Implementation leakage
• ~ Metadata is *intended state* not actual state
$ dmv find kctest1 -f json

[{
    "Category": "kctest1",
    "Blacklist Threshold": "1GB",
    "Encryption": "Yes",
    "Importance": "normal",
    "MailTo": "",
    "MaxRate": "1MB",
    "Modified": "2018-10-08T09:00:17",
    "Modified By": "security_oncall",
    "Oncall": "scribe_oncall",
    "Owner": "kcb",
    "Retention": 1,
}]

Customer Metadata
Scribe Categories
Customer Metadata

Scribe Categories

• + Know who our customers are
• - Implementation leakage
• ~ Metadata is *intended state* not actual state
• ~ Clear expectations?
$ dmv find kctest1 -f json

[{
  "Category": "kctest1",
  "Blacklist Threshold": "1GB",
  "Encryption": "Yes",
  "Importance": "normal",
  "MailTo": "",
  "MaxRate": "1MB",
  "Modified": "2018-10-08T09:00:17",
  "Modified By": "security_oncall",
  "Oncall": "scribe_oncall",
  "Owner": "kcb",
  "Retention": 1,
}]

Customer Metadata
Scribe Categories
Customer Metadata

Scribe Categories

- + Know who our customers are
- - Implementation leakage
- ~ Metadata is *intended state* not actual state
- ~ Clear expectations?
- + Customer data for operations
$ dmv find kctest1 -f json
[
{
   "Category": "kctest1",
   "Blacklist Threshold": "1GB",
   "Encryption": "Yes",
   "Importance": "normal",
   "MailTo": "",
   "MaxRate": "1MB",
   "Modified": "2018-10-08T09:00:17",
   "Modified By": "security_oncall",
   "Oncall": "scribe_oncall",
   "Owner": "kcb",
   "Retention": 1,
}
]
Customer Metadata

Scribe Categories

• + Know who our customers are
• - Implementation leakage
• ~ Metadata is *intended state* not actual state
• ~ Clear expectations?
• + Customer data for operations
• + Change history
$ dmv find kctest1 -f json
[
  {
    "Category": "kctest1",
    "Blacklist Threshold": "1GB",
    "Encryption": "Yes",
    "Importance": "normal",
    "MailTo": "",
    "MaxRate": "1MB",
    "Modified": "2018-10-08T09:00:17",
    "Modified By": "security_oncall",
    "Oncall": "scribe_oncall",
    "Owner": "kcb",
    "Retention": 1,
  }
]
Customer Metadata

Scribe Categories

• + Know who our customers are
• - Implementation leakage
• ~ Metadata is *intended state* not actual state
• ~ Clear expectations?
• + Customer data for operations
• + Change history
• - Implicit offerings create implicit expectations
Customer Metadata
Scribe Categories

How can we make this better?
• Manage intended -> actual state
Customer Metadata
Convergence & Failure

Intended state vs. actual state -> convergence
Customer Metadata
Convergence & Failure

Intended state vs. actual state -> convergence

"Blacklist Threshold": "1GB",
"MaxRate": "1MB",
Intended state vs. actual state -> convergence

"Blacklist Threshold": "1GB",
"MaxRate": "1MB",

# system aggregate rate limits vs. capacity
impossible desires + naïve guardrails =
Intended state vs. actual state -> convergence

"Blacklist Threshold": "1GB",
"MaxRate": "1MB",

# system aggregate rate limits vs. capacity
impossible desires + naïve guardrails = converge on system failure
**Customer Metadata**

**Convergence & Failure**

Intended state vs. actual state -> convergence

```
"Blacklist Threshold": "1GB",
"MaxRate": "1MB",
```

# system aggregate rate limits vs. capacity
impossible desires + naïve guardrails = converge on system failure
impossible desires + safe limits =
Customer Metadata
Convergence & Failure

Intended state vs. actual state -> convergence

"Blacklist Threshold": "1GB",
"MaxRate": "1MB",

# system aggregate rate limits vs. capacity
impossible desires + naïve guardrails = converge on system failure
impossible desires + safe limits = converge on customer failure
Customer Metadata
Convergence & Failure

Intended state vs. actual state -> convergence

"Blacklist Threshold": "1GB",
"MaxRate": "1MB",

# system aggregate rate limits vs. capacity
impossible desires + naïve guardrails = converge on system failure
impossible desires + safe limits = converge on customer failure
reasonable desires + safe limits + unexpected failure =
Customer Metadata
Convergence & Failure

Intended state vs. actual state -> convergence

"Blacklist Threshold": "1GB",
"MaxRate": "1MB",

# system aggregate rate limits vs. capacity
impossible desires + naïve guardrails = converge on system failure
impossible desires + safe limits = converge on customer failure
reasonable desires + safe limits + unexpected failure = ?
Customer Metadata

Scribe Categories

How can we make this better?

• Manage intended -> actual state
• Clarify expectations
$ dmv find kctest1 -f json
[
{
  "Category": "kctest1",
  "Blacklist Threshold": "1GB",
  "Encryption": "Yes",
  "Importance": "normal",
  "MailTo": "",
  "MaxRate": "1MB",
  "Modified": "2018-10-08T09:00:17",
  "Modified By": "security_oncall",
  "Oncall": "scribe_oncall",
  "Owner": "kcb",
  "Retention": 1
}
]
Customer Metadata

Expectations

- Tasks – Internal task ticketing system
- Tasks have priorities
- UBN = UnBreak Now!
- UBNs page the owner automatically
Customer Metadata

Expectations

- Organizationally meaningful priorities
- External accountability
- Enable better emergency response
Customer Metadata

Scribe Categories

How can we make this better?

- Manage intended -> actual state
- Clarify expectations
- Support implementation changes
Customer Metadata
Auditing Pattern

- “How do I turn on something new?”
$ dmv find kctest1 -f json

[{
  "Category": "kctest1",
  "Blacklist Threshold": "1GB",
  "Encryption": "Yes",
  "Importance": "normal",
  "MailTo": "",
  "MaxRate": "1MB",
  "Modified": "2018-10-08T09:00:17",
  "Modified By": "security_oncall",
  "Oncall": "scribe_oncall",
  "Owner": "kcb",
  "Retention": 1,
}]

Customer Metadata
Auditing Pattern
Customer Metadata

Auditing Pattern

• Goal: 100% encryption by default

• Challenges:
  • Encryption is a new backend feature
  • Encryption requires client upgrade, credential distribution
  • Fail open/closed?
Customer Metadata

Auditing Pattern

• Goal: 100% encryption by default

impossible desires + naïve guardrails = converge on system failure
impossible desires + safe limits = converge on customer failure
reasonable desires + safe limits + unexpected failure = ?
Customer Metadata

Auditing Pattern

• Goal: 100% encryption by default

impossible desires + naive guardrails = converge on system failure
impossible desires + safe limits = converge on customer failure
reasonable desires + safe limits + unexpected failure = ?

change(desires + limits) + failure = ?
Customer Metadata

Auditing Pattern

• Goal: 100% encryption by default
• Process:
  • Mass migration (“One perfect moment”)
Customer Metadata

Auditing Pattern

• Goal: 100% encryption by default

• Process:
  • Mass migration (“One perfect moment”)
  • Prepare then migrate (“Big list”)
**Customer Metadata**

Auditing Pattern

- Goal: 100% encryption by default
- Process:
  - Mass migration ("One perfect moment")
  - Prepare then migrate ("Big list")
  - Continuous auditing ("TDD for operations")
Customer Metadata
Auditing Pattern

• Goal: 100% encryption by default

• Effective auditing
  • Check metadata – is encryption enabled?
  • Check dependencies – are dependencies ready for encryption?
  • Check implementation – is category actually encrypted?
Customer Metadata

Customer Input
Customer Metadata

Customer Input

```bash
$ dmv find kctest1 -f json
[
    {
        "Category": "kctest1",
        "Blacklist Threshold": "1GB",
        "Encryption": "Yes",
        "Importance": "normal",
        "MailTo": "",
        "MaxRate": "1MB",
        "Modified": "2018-10-08T09:00:17",
        "Modified By": "security_oncall",
        "Oncall": "scribe_oncall",
        "Owner": "kcb",
        "Retention": 14,
    }
]
```
Customer Metadata

Customer Input

• Consistent public messaging
  • Group post?
  • Regular cadence
Customer Metadata
Customer Input

• In-person conversations
  • Gather allies
  • Address complexity upfront
  • Canary for automation
Customer Metadata

Customer Input

- Individual automated messaging
  - Be concise and link to additional documentation
  - Make it actionable
  - You’ll be wrong no matter what
  - Not every change is better for everyone
Customer Metadata

Customer Input

- Consistent public messaging (group posts)
- In-person conversations
- Individual automated messaging (tasks/tickets)
SLAs

You already have one
SLAs
You already have one

- If you don’t have an SLA your SLA is whatever the customer wants
SLAs
You already have one

- If you don’t have an SLA your SLA is whatever the customer wants
- The SLA is about expectations
SLAs
You already have one

• If you don’t have an SLA your SLA is whatever the customer wants
• The SLA is about expectations
• Expectations go both ways
SLAs
Zookeeper customers

- Zookeeper oncall gets UBNs for ensembles in trouble
SLAs
Zookeeper customers

- Zookeeper oncall gets UBNs for ensembles in trouble
  - Hardware failure? Bad deployment?
SLAs
Zookeeper customers

- Zookeeper oncall gets UBNs for ensembles in trouble
  - Hardware failure? Bad deployment?

filter(zk\global\.\((0[389]|29|4[2-8]|6[589]|72|103))
SLAs

Zookeeper customers

- Zookeeper oncall gets UBNs for ensembles in trouble
  - Hardware failure? Bad deployment?
  - Customer load?

\texttt{filter(zk\.global\.(0[389]|29|4[2-8]|6[589]|72|103))}
SLAs
Zookeeper customers

- Zookeeper oncall gets UBNs for ensembles in trouble
  - Hardware failure? Bad deployment?
  - Customer load?
- Manual alarm triaging is a symptom
SLAs
Zookeeper customers

- Zookeeper oncall gets UBNs for ensembles in trouble
  - Hardware failure? Bad deployment?
  - Customer load?
- Manual alarm triaging is a symptom
  - The system can’t defend itself from bad actors
SLAs
Zookeeper customers

- Zookeeper oncall gets UBNs for ensembles in trouble
  - Hardware failure? Bad deployment?
  - Customer load?
- Manual alarm triaging is a symptom
  - The system can’t defend itself from bad actors
  - We don’t have metadata or we’re not using it
SLAs

Zookeeper customers

- Zookeeper oncall gets UBNs for ensembles in trouble
  - Hardware failure? Bad deployment?
  - Customer load?
- Manual alarm triaging is a symptom
  - The system can’t defend itself from bad actors
  - We don’t have metadata or we’re not using it
  - Fear of conflict or visibility
What do we need?

- Written SLA
- Expectations go both ways
- Problems are solved by the right team
- Published metrics
Monitoring

The p100 problem
Monitoring

The p100 problem

- Monitoring is part of the service
Monitoring

The p100 problem

- Monitoring is part of the service
- Is 99% availability good?
Monitoring
The p100 problem

• Monitoring is part of the service
• Is 99% availability good?
  • 100/10000 servers failing chef runs
Monitoring
The p100 problem

- Monitoring is part of the service
- Is 99% availability good?
  - 100/10000 servers failing chef runs
  - 1/100 database masters failing chef runs
Monitoring

The p100 problem

- Monitoring is part of the service
- Is 99% availability good?
  - 100/10000 servers failing chef runs
  - 1/100 database masters failing chef runs
  - 1/100 zookeeper ensembles unavailable
**Monitoring**

*The p100 problem*

- Monitoring is part of the service
- Is 99% availability good?
  - 100/10000 servers failing chef runs
  - 1/100 database masters failing chef runs
  - 1/100 zookeeper ensembles unavailable
  - 10/1000 scribe categories failing writes
Monitoring
The p100 problem – chef monitoring

- Chef team
  - Chef backend infrastructure (is the service up)
  - Global run success (is chef working for customers)
Monitoring
The p100 problem – chef monitoring

- Chef team
  - Chef backend infrastructure (is the service up)
  - Global run success (is chef working for customers)

- Customer teams
  - Per-customer run success
Monitoring
The p100 problem – chef monitoring

• Chef team
  • Chef backend infrastructure (is the service up)
  • Global run success (is chef working for customers)

• Customer teams
  • Per-customer run success

• Sane defaults + flexibility
  • Tunable thresholds (mandatory minimums)
Monitoring
The p100 problem – chef monitoring

• Chef team
  • Chef backend infrastructure (is the service up)
  • Global run success (is chef working for customers)

• Customer teams
  • Per-customer run success

• Sane defaults + flexibility
  • Tunable thresholds (mandatory minimums)
  • Configurable notifications
Monitoring
The p100 problem – chef monitoring

- Chef team
  - Chef backend infrastructure (is the service up)
  - Global run success (is chef working for customers)

- Customer teams
  - Per-customer run success

- Sane defaults + flexibility
  - Tunable thresholds (mandatory minimums)
  - Configurable notifications
  - Automatic dependencies
**Monitoring**

The p100 problem – chef monitoring

What do we need?

- Monitoring of our service
- Monitoring as a service
Additional complexity

Everything was going so well
Additional complexity
Lifecycle - Decommissioning

- zookeeper – what if an ensemble becomes unused?
- scribe – what if a category becomes unused?
- What does unused mean?
- Would you be able to tell?
Additional complexity

Customers with customers
Additional complexity

Customers with customers

- Metadata service load
Additional complexity

Customers with customers

- Metadata service load
- Customers blaming their customers
Additional complexity
Customers with customers

- Metadata service load
- Customers blaming their customers
  - Incidents
  - Ownership
  - Monitoring
  - Capacity
Conclusions
Service Maturity Goals and Tips

Config Smell

- Separate customer metadata from service implementation
- Define scalable service levels
Service Maturity Goals and Tips

Customer Metadata

• Know who your customers are
• Define expectations for success and failure (convergence)
• Use organizationally meaningful data (task priorities)
• Plan for future changes (auditing pattern)
• Automated tasks are great (for irritating colleagues)
Service Maturity Goals and Tips

SLAs and Monitoring

• If you don’t have an SLA your SLA is whatever the customer wants
• Expectations and accountability go both ways
• Monitoring is part of the service you offer
Service Maturity Goals and Tips

Additional Complexity

• Manage the whole lifecycle

• Your customers will build services out of your service
Service Maturity Goals and Tips

Final thoughts

• There is no one right answer
• You don’t get to pick where to start
• You do get to decide what your service is and what it isn’t
• Leave things better than you found them
facebook  Thank you