MANAGING YOUR HEROES
The People Aspect of Monitoring
(a.k.a. Dealing with Outages and Failures)

Alex Solomon
alex@pagerduty.com
WHO AM I?

Alex Solomon

• Founder / CEO of PagerDuty
• Intersect Inc.
• Amazon.com
DEFINITIONS
Service Level Agreement (SLA)

Mean Time To Resolution (MTTR)

Mean Time To Response

Mean Time Between Failures (MTBF)
Can we prevent them?

"I'll have an ounce of prevention."
PREVENTING OUTAGES

- Single Points of Failure (SPOFs) [X]
- Redundant systems [✓]
- Complex, monolithic systems [X]
- Service-oriented architecture [✓]
Netflix distributed SOA system
PREVENTING OUTAGES

Change

(not much you can do about this one)
OUTAGES

SHIT HAPPENS
FAILURE LIFECYCLE
Monitor -> detect failure -> Alert -> Investigate -> Fix -> Root-cause Analysis
Critical Incident Timeline

<table>
<thead>
<tr>
<th>Alert</th>
<th>Investigate</th>
<th>Fix</th>
</tr>
</thead>
<tbody>
<tr>
<td>Issue is detected</td>
<td>Engineer starts working on issue</td>
<td>Issue is fixed</td>
</tr>
</tbody>
</table>

**RESPONSE TIME**

**RESOLUTION TIME**
MONITOR
MONITOR EVERYTHING!

All levels of the stack

- Data center
- Network
- Servers
- Database
- Application
- Website
- Business Metrics
WHY MONITOR EVERYTHING?

Metrics!

Metrics!

Metrics!
TOOLS

• Internal monitoring (behind the firewall):
  • Nagios
  • splunk

• External monitoring (SaaS-based):
  • New Relic
  • pingdom

• Metrics:
  • Graphite or Datadog
ALERT
Best Practice: Categorize alerts by severity.
SEVERITIES

Define severities based on business impact:

• **sev1** - large scale business loss
• **sev2** - small to medium business loss
• **sev3** - no immediate business loss, customers may be impacted
• **sev4** - no business loss, no customers impacted

2 critical severities

2 non-critical severities
Each severity level should have its own standard operating procedure (SOP):

- **Who**
- **How**
- **Response time**
• **Sev1**: Major outage, all hands on deck
  • Notify the entire team via phone and SMS
  • Response time: 5 min

• **Sev2**: Critical issue
  • Notify the on-call person via phone and SMS
  • Response time: 15 min

• **Sev3**: Non-critical issue
  • Notify the on-call person via email
  • Response time: next day during business hours
• **Sev 1** incidents
  
  • Rare
  
  • Rarely auto-generated
  
  • Frequently start as sev 2 which are upgraded to sev 1
• **Sev2** incidents
  
  • More common
  
  • Mostly auto-generated
• **Sev3** incidents
  
  • Non-critical incidents
  
  • Can be auto-generated
  
  • Can also be manually generated
• Severities can be **downgraded** or **upgraded**

• ex. **sev2 → sev1** (problem got worse)

• ex. **sev1 → sev2** (problem was partially fixed)

• ex. **sev2 → sev3** (critical problem was fixed but we still need to investigate root cause)
One more best-practice:

Alert *before* your systems fail completely
Main benefit of severities

Only page on critical issues (sev1 or 2)
Preserve sanity
Avoid “Peter and the wolf” scenarios
ON-CALL BEST PRACTICES

Person Level

Team Level
ON-CALL AT THE PERSON LEVEL

Cellphone
Cellphone

Smart phone

OR

AND
4G / 3G internet

4G hotspot  4G USB modem  3G/4G tethering

(don’t forget your laptop)
• **Time zero**: email and SMS
• **1 min later**: phone-call on cell
• **5 min later**: phone-call on cell
• **5 min later**: phone-call on landline
• **5 min later**: phone-call to girlfriend
Bonus: vibrating bluetooth bracelet
ON-CALL AT THE TEAM LEVEL

**Rarely** send alerts to the entire team

**sev1** OK

**sev2** NO
On-call schedules:

• *Simple rotation-based* schedule
  • ex. weekly - everyone is on-call for a week at a time
• Set up a *follow-the-sun* schedule
  • people in multiple timezones
  • no night-shifts
What happens if the on-call person doesn’t respond at all?
If you care about uptime, you need **redundancy** in your on-call.
Set up multiple on-call levels with automatic \textbf{escalation} between them:

\textbf{Level 1}: Primary on-call

\textbf{Level 2}: Secondary on-call

\textbf{Level 3}: Team on-call (alert entire team)
Best Practice: Put *management* in the on-call chain

**Level 1**: Primary on-call

*Escalate after 15 min*

**Level 2**: Secondary on-call

*Escalate after 20 min*

**Level 3**: Team on-call (alert entire team)

*Escalate after 20 min*

**Level 4**: Manager / Director
Best Practice: put *software engineers* in the on-call chain

- Devops model
- Devs need to own the systems they write
- Getting paged provides a *strong incentive* to engineer better systems
Best Practice: measure **on-call performance**

“You can’t improve what you don’t measure.”

- Measure: mean-time-to-response
- Measure: % of issues that were escalated
- Set up policies to encourage good performance
  - Put managers in on-call chain
  - Pay people extra to do on-call
Network Operations Center
NOC with lots of Nagios goodness
NOCs:

• Reduce the mean-time-to-response drastically
• Expensive (staffed 24x7 with multiple people)
• Train NOC staff to fix a good %age of issues
• As you scale your org, you may want a hybrid on-call approach (where NOC handles some issues, teams handle other issues directly)
Critical Incident Timeline

**Alert**
- Issue is detected
- Alerting system gets ahold of somebody

**Investigate**
- Engineer starts working on issue
- Engineer gets to a computer, connects to internet
- Engineer is aware of issue

**Fix**
- Issue is fixed
- Engineer starts working on issue
- Issue is fixed

**RESPONSE TIME**
- Alert
- Investigate

**RESOLUTION TIME**
- Investigate
- Fix
### Alert

<table>
<thead>
<tr>
<th>Alerting system gets ahold of somebody</th>
<th>Engineer gets to a computer, connects to internet</th>
</tr>
</thead>
<tbody>
<tr>
<td>Issue is detected</td>
<td>Engineer is aware of issue</td>
</tr>
<tr>
<td></td>
<td>Engineer starts working on issue</td>
</tr>
</tbody>
</table>

#### How to minimize:

- Alert via phone & SMS
- Alert multiple times via multiple channels
- Failing that, escalate!
- Failing that, escalate to manager!

#### How to minimize:

- Carry 4G internet device + laptop at all times
- Set loud ringtone at night
RESEARCH & FIX
How do we reduce the amount of time needed to investigate and fix?
Set up an Emergency Ops Guide:

- When you encounter a new failure, document it in the Guide
- Document *symptoms, research steps, fixes*
- Use a *wiki*
Emergency Ops Guide

- DB Primary Failure
  - Symptoms
  - Procedure
- Account Lock Pileups
  - Symptoms
  - Procedure
- Frontend Failure
  - Symptoms
  - Procedure
- CDN (CacheFly) Failure
- Strange IP reallocations on the frontend
  - Symptoms
  - Procedure
- Email Queue Overflow
  - Symptoms
  - Procedure
- Missing BG Task for Notification
  - Symptoms
  - Procedure
- Restarting a host
- Bad Phone Number
- MySQL replication lag
  - Symptoms
  - Procedure
- Phone/SMS/Email Provider Failure
  - Symptoms
  - Procedure
- Multi Provider Failure Button
  - Symptoms
  - Procedure
- AWS Fails
  - Symptoms
DB Primary Failure

Symptoms

- Can't log into [REDacted]
- Getting 5xx from all the app pages for no clear reason
- Error messages or logs indicate problems connecting to the DB

Procedure

- If the primary DB machine (db.pagerduty.com) is accessible, log in and spend a minute or so to see if you can solve the problem directly.
- If the machine is unreachable, or if you can't solve the problem, do a [DRBD Flip]
Automate fixes

or

Add more fault tolerance
You need the right tools:

- Tools to help you diagnose problems faster
- Comprehensive monitoring, metrics and dashboards
- Tools that help search for problems in log files quickly (ie. Splunk)
- Tools to help your team communicate efficiently
  - Voice: Conference bridge, Skype, Google Hangout
  - Chat: Hipchat, Campfire
Best Practice: Incident Commander
Incident Commander:

- Essential for dealing with severe issues
- In charge of the situation
  - Providers leadership, prevents analysis paralysis
  - He/she directs people to do things
  - Helps save time making decisions
Questions?

Alex Solomon
alex@pagerduty.com