Heroku Emergency Strategy: Incident Command System and 8 Hour Ops Rotations for Fresh Minds

Wednesday, April 27, 2011 at 8:35AM

Todd Hoff in Strategy, amazon

In Resolved: Widespread Application Outage, Heroku tells their story of how they dealt with the Amazon outage. While taking 100% responsibility for the downtime, they also shared a number of the strategies they used to bring their service back to full working order.

One of Heroku's most interesting strategies wasn't a technical hack at all, but how they consciously went about deploying their Ops personnel in response to the emergency. An outline of their strategy is:

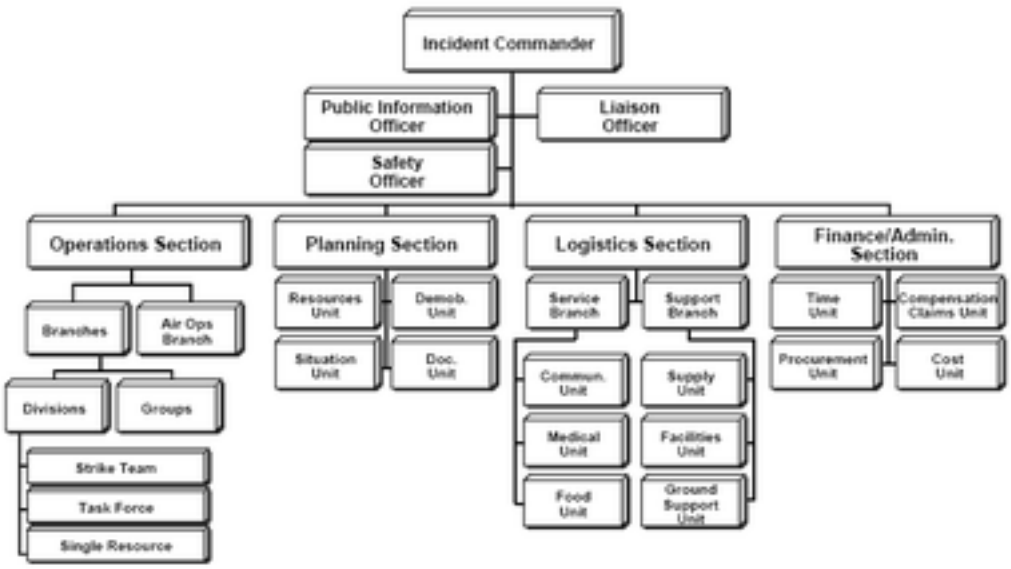
- Monitoring systems immediately alerted Ops to the problem.
- An on-call engineer applied triage logic to the problem and classified it as serious, which caused the on-call Incident Commander to be woken out of restful slumber.
- The IC contacted AWS. They were in constant contact with their

AWS representative and worked closely with AWS to solve problems.

- The IC alerted Heroku engineers. A full crew: support, data, and other engineering teams worked around the clock to bring everything back online.
- The Ops team instituted an emergency incident commander rotation of 8 hours per shift, once it became clear the outage was serious, in-order to keep a fresh mind in charge of the situation at all times.
- Status updates were frequentish, even if they didn't add a lot of new information, just to let everyone know Heroku was working the problem.

Incident Command System

Chrishenn in a comment on the Heroku post on Hacker News, thinks Heroku was using an emergency response system based on the Incident Command System model: a systematic tool used for the command, control, and coordination of emergency response. A picture of the model from wikipedia:



I've never heard of ICS before, but it looks worth looking into if you are

searching around for a proven structure. Chrishenn says it works:

> *I've experienced it first hand and can say it works very well, but I have never seen it used in this context. The great thing about it is it's expandability---it will work for teams of nearly any size. I'd be interested in seeing if any other technology companies/backend teams are using it.*

Lessons Learned

- Have a monitoring system so you know immediately when there are problems.
- Be a really big customer so Amazon will help you specifically with your problems. This seemed to help Heroku a lot. I noticed in the Amazon developer forums a lot of people forgot to do this and didn't get the personal help they needed.
- Have a structured emergency response process. Heroku has an on-call engineer, an IC coordinator, an escalation process, and the ability to call in all the troops when needed. It sounds like Heroku handled the incident like a well oiled machine. That takes practice, so practice ahead of failures instead of trying to figure out all this stuff when the EBS is falling.
- Recognizing that tired people make bad decisions and putting the coordinator on 8 hour rotation to prevent this type of secondary failure is really genius.
- Heroku's status update policy shows an insightful understanding of customer psychology. Managing customer feelings and expectations was a great move. Nobody wants to feel abandoned in a pressure situation. Even if the status updates did not contain a lot of details, they did show Heroku was there and cared.
- Do not over promise and under deliver. Heroku's status messages

were measured, reassuring, and honest. Heroku didn't know what was wrong or when the service would be up again, so they just didn't say. Nothing creates resentment faster than lying in these situations.

Related Articles

- Hacker News Thread
