DevOps and Automation

Benjamin Brewster & Elijah Voigt

Why You Need to Care

- Deploy the same configuration a billion times
- Deploy the same configuration a billion times *right now*
- Need to convince your boss that developers are really good at taking care of the systems that they wrote
- Design a back-end that runs effortlessly, with few people, at cloud-scale

- Note: there aren't any hands-on portions to this lecture

DevOps Defined

- This trendy buzzword can mean different things:
  - You write software and maintain it because Management is too cheap to hire IT administrators
  - You write services and wear the pager for when they go down
  - You spend half your time programmin' and half your time wranglin'
  - Who knows but here's $120K

DevOps Defined

- "DevOps is the codification and automation of computing infrastructure"

- Lets break it down:
  - Codification: precise specification of what we're doing and with what; we're not just adding features randomly: we have an objective we're trying to meet (more later)
  - Automation: we want this to build, run, size, and heal itself
  - Computing Infrastructure: server and networking hardware, services software, and the telemetry hardware and software that monitors it all

The Dream aka Nightmare

- You woke up this morning and *BAM* you're a sysadmin at Farcebook!
- You've got thousands of servers all over the world, all running different OSs and providing different services
- How do you micro-manage these, and do you even want to? If something's wrong with the configuration of a particular machine, do you even care?
- If you want to push an update, how do you do that to all of them? And can you really type a few hundred commands thousands of times?

The Woken-Up Reality is Not Much Better

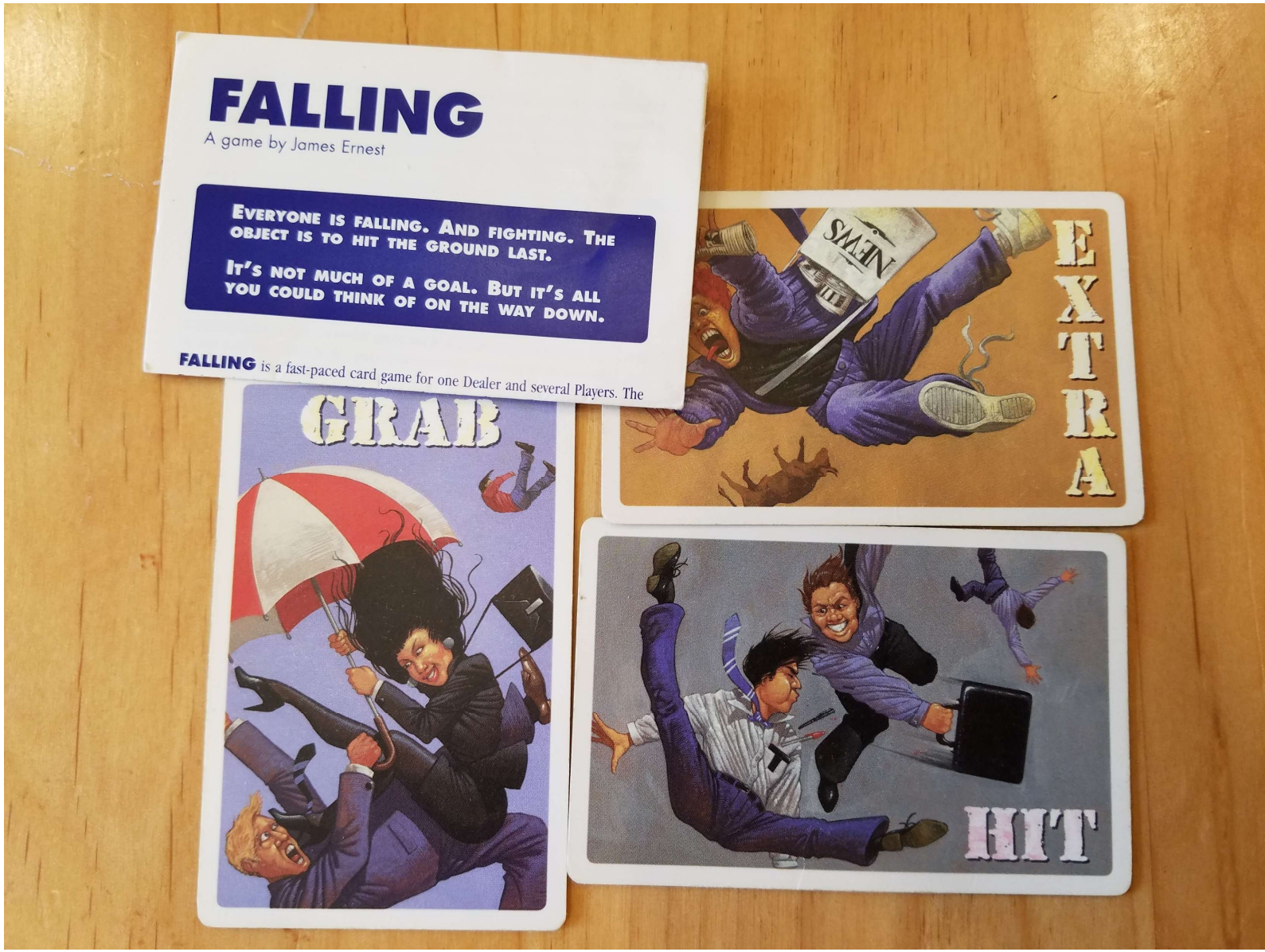
- Your real-life cat .GIF streaming service just took off!
- It's on the Linux box under your desk… but now it needs to be running on a dozen servers online ASAP
- Do you even remember how you got it working originally? Is the specific configuration all that important?

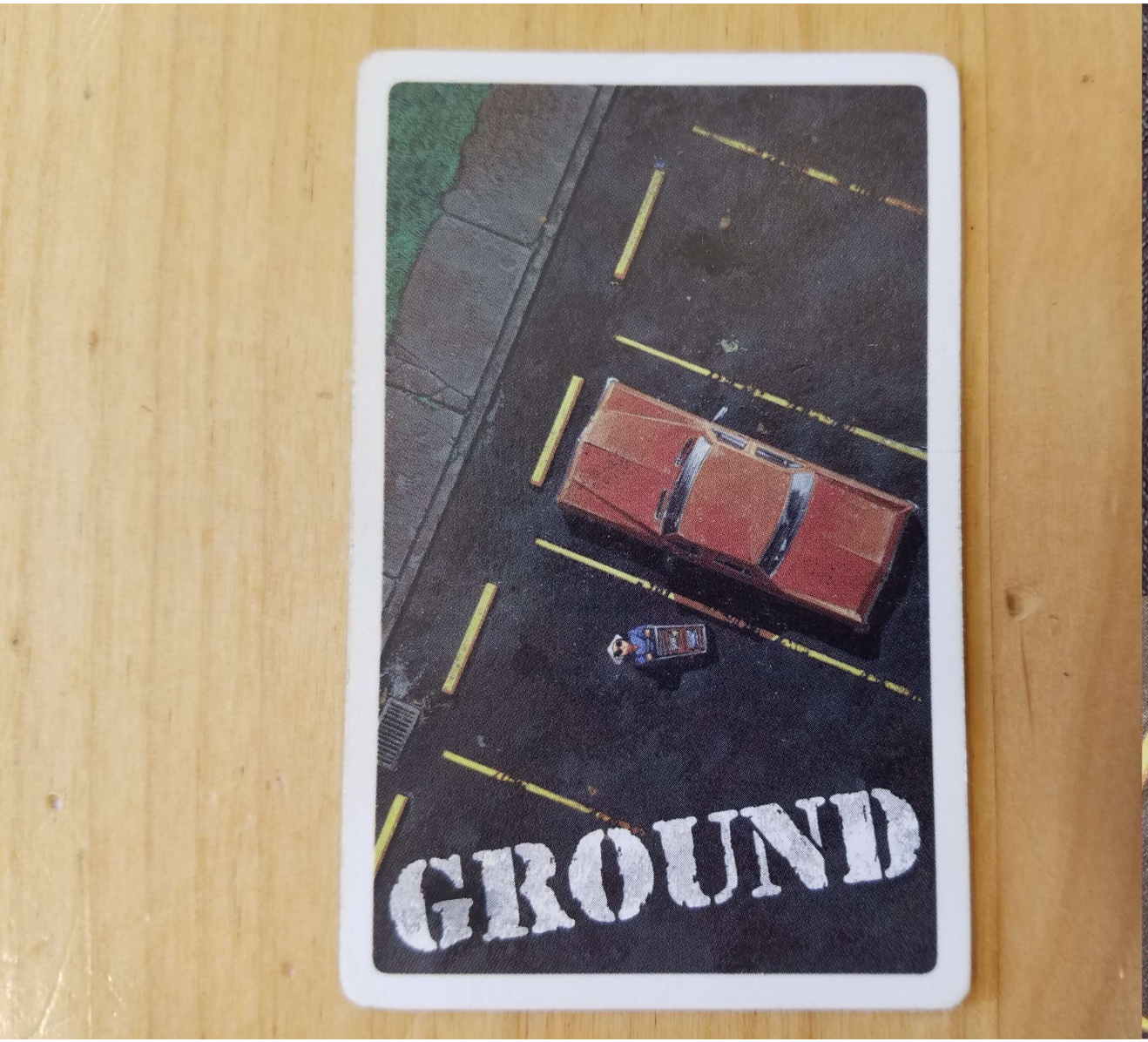
- Your inbox is flooding with complaints that their cat streams aren't .GIFing hard enough!

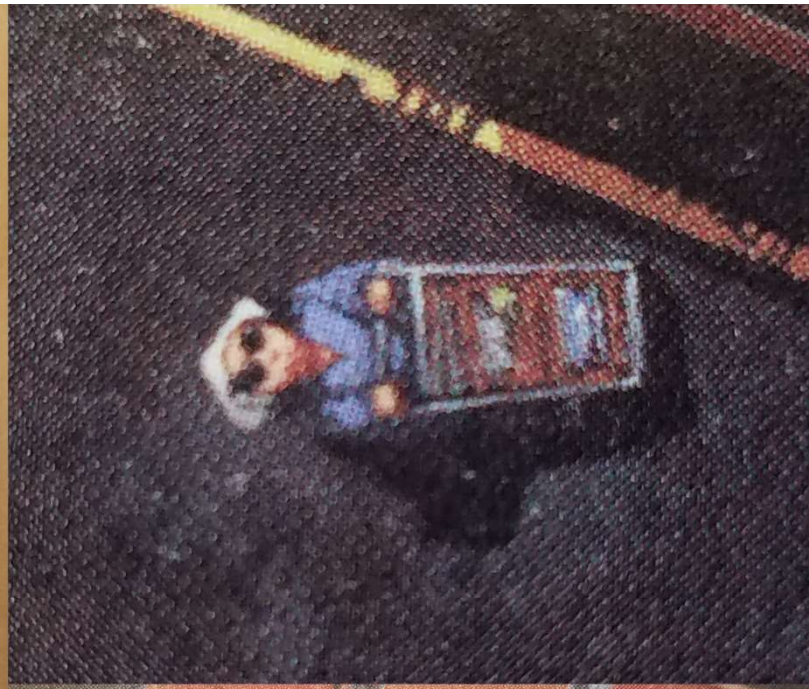
Ephemerality

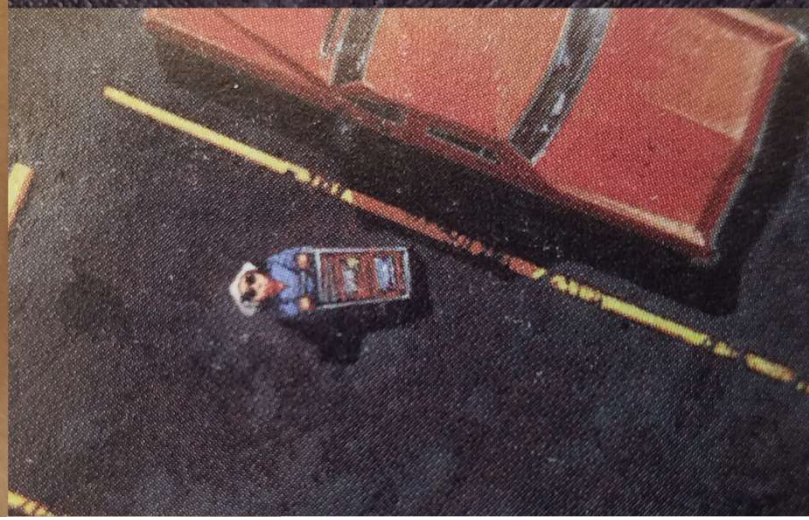
- This whole class, we have talked about this new concept of purely ephemeral configurations:
  - VMs get shnapshotted, get restored back to previous times, get spun up and spun down, and never even know what hardware they're on
  - Containers don't even get the decency of booting themselves, and have to assign themselves random, meaningless names out of pity before they are shut down, and that not even gently
- *Do you even remember how you got it working originally? Is the specific configuration all that important?*

- Not even a *hint* of why this is happening, or where, or for what reason, but these processes immediately spring into (violent) action for as long as they can, until…




GROUND





Configuration Management

- Configuration Management is the ultimate expression of ephemerality
- Configuration Management means that the attributes, configuration, and job duties of a particular system or service are kept as a text file that can be distributed as needed:
  - To build a new node, for whatever reason (expansion, surge in load, etc.)
  - To repair an old node, for whatever reason
- In the old days, skilled greybeards knew the reason driver X crashes in phase Y of the moon with Z software installed
- Nowadays, we *do not care:* burn it down and build it again

Configuration Management At the Core

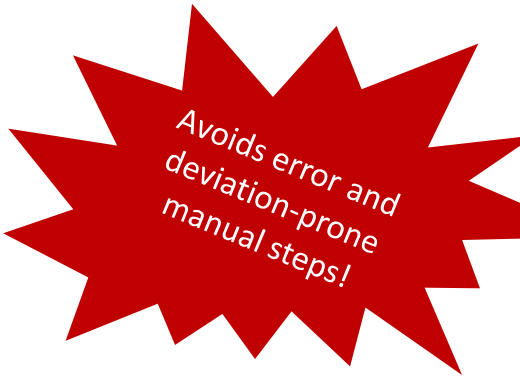
- Configuration Management is the core of DevOps
- It is the use of configuration files, scripts, and CM services to automate the management of your computers, VMs, and servers

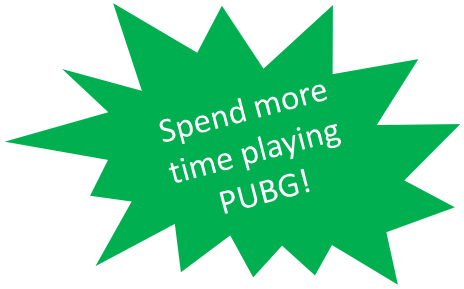
- For example, we *really want* to just do this:
  1. Spin up a Linux computer
  2. Install the dependencies for the catsGifsNow service software
  3. Download the catGifsNow Git repository's source code
  4. Build catGifsNow
  5. Start the catGifsNow service
  6. Add this instance of catGifsNow to our load balancer

Configuration Management At the Core

- Configuration Management is the core of DevOps
- It is the use of configuration files, scripts, and CM services to automate the management of your computers, VMs, and servers

- For example, we *really want* to just do this:
  1. Spin up a Linux computer
  2. Install the dependencies for the catsGifsNow service software
  3. Download the cagGifsNow Git repository's source code
  4. Build catGifsNow
  5. Start the catGifsNow service
  6. Add this instance of catGifsNow to our load balancer





Configuration Management to the Rescue

- With Configuration Management, we *codify* these instructions (which are too complex for a shell script), and then hand them to a program
- We tell that program: "Make these X computers arrive at Y state"

- Examples of common CM and CM-like programs:
  - Chef
  - Puppet
  - Continuum and other Managed Service Provider middleware
  - Ansible

- Let's compare and contrast these!

Chef and Puppet

- In Chef and Puppet, you install a daemon onto the target system which does these things:
  - Tracks the host state (packages, files, network connections, etc.)
  - Receives updates from the Controller
  - Polls for updates every half-hour or so
- Thus, these operate on a "pull" model: the installed agent checks for updates periodically, then performs those updates
- In Chef, you write Ruby scripts that define host configuration
- With Puppet, you use the proprietary Puppet config language
- Ben's opinion is that installing daemons is a drag because they take space, seem to always be crashing themselves, and have to be installed in the first place, which seems counter to the whole idea

MSP Middleware

- Some companies offer subscriptions for agents that do host monitoring and configuration, all conveniently controlled from a web console
- Common features:
  - Remote access via both KVM and file transfer
  - Deploying packages for installation, scripts for execution
  - Virus protection
  - System update management
  - Configurable system alerts (server down, etc.)
- Works very well for very distributed sites, sets up a very turn-key MSP that can run with limited personnel
- …but requires an installed (and in my experience buggy) agent, with constant high monthly costs to a third party, and isn't really CM
- If you have an IT team, you won't be using MSP software in any event

Ansible

- Ansible operates on a "push" model: whenever you want, you can execute commands on the hosts
  - Can be configured to download command sets periodically, making this a "pull"
- This is the mechanism, straight from the documentation: "Ansible works by connecting to your nodes and pushing out small programs, called "Ansible modules" to them. These programs are written to be resource models of the desired state of the system. Ansible then executes these modules (over SSH by default), and removes them when finished."
- I.e., it establishes an SSH connection, pushes over the modules it needs to arrive at the state you've told it to assume, then runs those commands on the pushed module; finally, removes the modules when done (which is good: they can't be used after the fact)

Ansible

- The push model is less efficient, because a controller has to evaluate the script given to it, decide on the modules needed, upload those modules, run the commands, then remove the modules

- A pull model simply has the agent run the commands it receives, and is therefore much more distributed

- Nevertheless, I prefer Ansible because I don't have to maintain an agent install! I've done that with other vendors, and hated it: it's always crashing, or takes too long to install, or we don't have the admin passwords, etc.

Ansible - Not a Shell Script

- Again, we're not writing a shell script, here: we're identifying a target state that the system should be in, and telling *Ansible* to get it there: *this* is Configuration Management
- When you need an additional MySQL server, you just tell Ansible:
  - "Hey, here's a blank computer, make it a MySQL server please"
- And it *does it*, without human error and much faster than a person could… so you sleep through the night

Ansible - Not a Shell Script

- Again, we're not writing a shell script, here: we're identifying a target state that the system should be in, and telling *Ansible* to get it there: *this* is Configuration Management
- When you need an additional MySQL server, you just tell Ansible:
  - "Hey, here's a blank computer, make it a MySQL server please"
- And it *does it*, without human error and much faster than a person could… so you sleep through the night
- Ansible is not a service or a daemon: you could just run the commands from a low-end laptop connected to the internet, and it'll reach out and make the changes you tell it to
- I.e., Ansible does not maintain state: it is not for monitoring

Ansible: Supported Controllers and End-Points

- The controlling software is easiest to install on Linux, but can be installed onto Windows with the new bash support
  - Can also use Cygwin, or a Linux VM as we will demo at our next lecture

- Supported end-point/host operating systems:
  - Windows, Mac, and POSIX (so, UNIX, Linux, BSD, macOS, etc.)

Goals of Automation

- Here are our objectives with Ansible:
  - **Remove human error:** when you tell a computer to do something, it does the exact same thing every single time
  - **Do things faster:** when you tell a computer to do something, it does it way faster than people
  - **Reproducibility:** when you can codify something, you can also test it. While testing might feel like a waste of time, it is very useful for verifying that "no matter what, these exact things *definitely* work" - and you only have to reach that stage once
  - **Auto-remediation:** We can tell computers how to solve simple problems: "Did this daemon die? Here's how to restart it. Send me a text if that doesn't work. Call me if it doesn't work 3 times in a row."

- Remember: Ansible isn't a monitoring platform: it's telling the *hosts* how to do these things!

DevOps Reviewed

- DevOPs people worry about SLI, SLO, SLA - they're not just programmers (more on this in week 10)
  - **Service Level Indicator:** A quantitative measurement of a particular facet of a service: uptime, number hosts contacted, patches installed per hour, etc.
  - **Service Level Objectives:** A target value or range of values for an SLI
  - **Service Level Agreement:** The contractual agreement that specifies what SLOs we're attempting to meet, and what happens when we meet, exceed, or fail to meet them
- This means: DevOps people don't just fix things, they write software to assure these objectives, and attempt to anticipate failure modes
- When things fail unexpectedly, they reconfigure the systems to be hardened against that particular failure mode

Conclusion

- DevOps pays a ton of money
- DevOps is a mindset, an approach to service or platform delivery
- DevOps can be confusing to talk about, as definitions vary
- OSU has amazing resources towards learning DevOps culture and methodology:
  - Linux Users Group
  - CASS
  - Open Source Lab (OSL)