Sample Talk Submission (40 mins)

Title
Scalable Meatfrastructure: Building Stable DevOps Teams

Proposal
Abstract
When we think of scalable infrastructure, we think of technologies like AWS, Kafka, or Mesos. But all these shiny things, at the end of the day, depend on people. You wouldn't rollout containerization while your app has a memory leak, so you shouldn't expand your DevOps team without addressing the leaks in your day-to-day communication and processes. I will address how to build stable teams by converting oral cultures to written ones, establishing standard processes for handling developer requests, and creating resilient onboarding practices so new employees can act quickly and independently. Come learn some of the steps necessary to make your most important platform--your engineers--stable and successful.

Talk Background
This talk will address common issues facing DevOps teams as they expand their headcount. In my two years at New Relic, our Site Engineering team grew from 5 to over 15 engineers, supporting an engineering organization of several hundred. We struggled with communication drift, oncall handoffs, poor onboarding experiences, and many other issues experienced through expansion. Although I'm not a manager, I was directly involved in fixing some of these issues and have seen our distributed team organize and stabilize for the better.

Outline
Opening Example: Your company is making money! Wow! You double your DevOps team, include hiring Shanna, a remote engineer. Shanna is experienced and competent, so even though she's only been to the main office once and you often forget to include her in standups, you figure she's ok. Several months go by and finance contacts you about the cost of some servers. You investigate and, for some reason, these servers are being used by one small service. Turns out these were set up by Shanna, who didn't realize the recent decision to put all new services on shared Docker hosts. That information was spoken in standups and partially written down in a document you can't find right now. No one thought to update her, and now you have to migrate this service and take up that development team's time. But it turns out Shanna setup some unique permissions on those servers, and you can't ask her about it until she comes online in 6 hours.

This could have all been avoided with the following considerations.

Enable logging:

- Before you can scale your team, you need to make sure your current one is stable.
    - Would you survive the Bus test?
- Write everything down (yes, everything)
    - Your team's mission, your team's chosen tools and methodologies, standard practices/processes (how do you attend a conference?), what broke last night
    - Make your Oral Culture a Written one instead. Empower everyone to ask their coworkers for docs.
- Do it: Pull a service into a test environment, break it, and have one engineer fix it and write a runbook of steps needed. You just saved a future oncall situation.

Absolute path:

- Writing it down makes no difference if 5 people write docs in 5 different places.
- You need a Single Source of Truth:
    - Everyone knows where it is
    - Everyone can contribute to it
    - It's searchable
    - It's archivable (team chat logs don't count)
- Do it: Take DevOps to heart and make an app that tracks the pain points in your infrastructure. You can call it What's On Fire Today? and any engineer can check there for ongoing issues. Great for oncall handoffs.

Default configuration:

- Your team needs to operate smoothly within your company architecture
- If a dev team is launching a new service, what happens? How do they communicate with you? What questions do you need to ask them?
- Evaluate your current architecture/processes, think about where you want it to be in a year, and Make a Decision.
    - Choose a process to apply to 90% of your architecture. This is now your standard. Allow room in your standard process for one-offs, but otherwise define a process and make it easy to follow.
- Do it: If you decide your infrastructure model will be EC2 instances running Rails apps and PostgresDB, make this known. Write documentation for developers and scripts for your Ops team, so provisioning and deploying is easy.

Stable distribution:

- Your local team is stable, but now it's time to stabilize your distributed engineers
- Remote workers are not second-class citizens.
    - Make standup notes accessible. Record meetings. Do video calls.
    - Fly them in at least once a quarter.
    - Make sure they get the same swag your local team does.
- Do it: Give the next big project to a remote engineer. This will not only increase their visibility but will expose the weak points in your documentation, communication, and shared processes.

Resilient under high load:

- It's time to expand your headcount and scale your team without draining current resources
- If your docs are unclear, the first commit process is out of date, it takes three engineers to remember all the proper logins, etc, your team will be slowed down to bring on one new person.
    - You can have the new hire update the docs as they go, but realize what this says about your process and priorities.
- Do it: Have a current engineer sit in a room and go through your onboarding process alone. They should be able to make their first commit following only the steps provided.

Conclusion

This talk is just the beginning. As long as you take the main principles to heart, you'll be in a good place to focus on scaling your technical infrastructure.