



# HOW TO PREVENT CATASTROPHIC DOOM ON YOUR NEXT FEDERAL PROJECT

*Presented by Ryan Kenney*



[ryan.kenney@coveros.com](mailto:ryan.kenney@coveros.com)



<https://www.linkedin.com/in/ryan-kenney-13703887>



<https://twitter.com/rkenney525>





# Overview

- Introduction
- Overview of Case Study
- Defining a Solution
- Fantastic Problems and How to Solve Them
- Wrap-up
- Q&A



# Who is this guy?

Ryan Kenney



- Born/raised in Rockingham, VA
- Senior Consultant at Coveros
- With Coveros since June 2013
- Agile Developer
- Technical Architect
- DevOps Tech Lead



## Overview of Case Study



# Case Study Overview

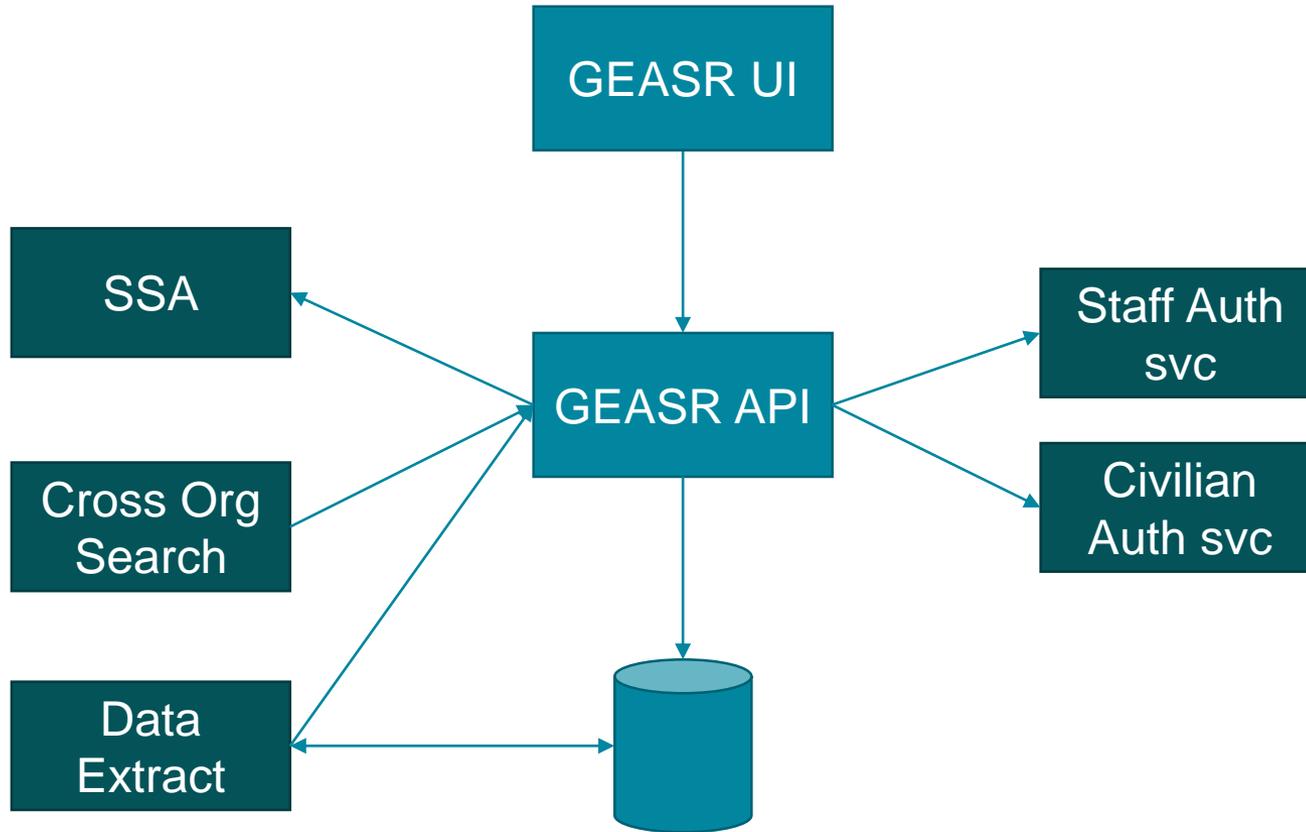
## ***GEASR 2.0***

The Government Enrollment and Application Support Radiator (**GEASR**) is the (fictitious) online system responsible for consolidating government program enrollment processes across organizations.

(Based on the experiences and interactions of select government projects)



# GEASR 1.0 In Action





# Case Study Overview

## What we have

- Monolithic application
- Uptime dependent on external services
- Legacy tools/frameworks
- Manual deployments
- Limited automated testing

## What we want

- Less internal coupling
- Less external coupling
- Modern (supported) tools
- Automated deployments
- Lots of automated testing



# Case Study Overview

## What we want

- Less internal coupling
- Less external coupling
- Modern (supported) tools
- Automated deployments
- Lots of testing

## How to get there

- Domain Driven Design and Microservices
- Smarter **health checks**
- Balance between **LTS** and **cutting edge**
- Automate Everything™
- Don't wait to write tests



# “Verticals and Horizontals”

	Microservice A	Microservice B	Microservice C
<i>Enterprise Architecture</i>			
Quality Assurance			
DevOps			
User Experience/Engagement			



## Defining a Solution

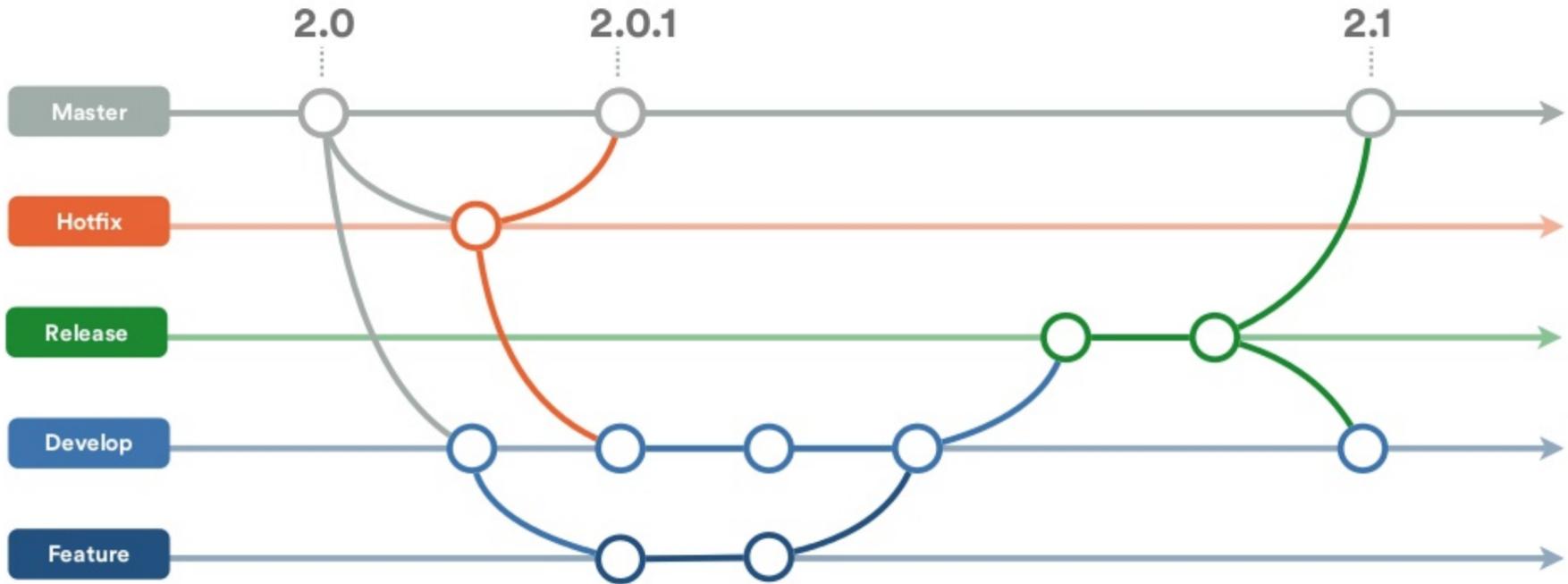


# What are we trying to deploy?

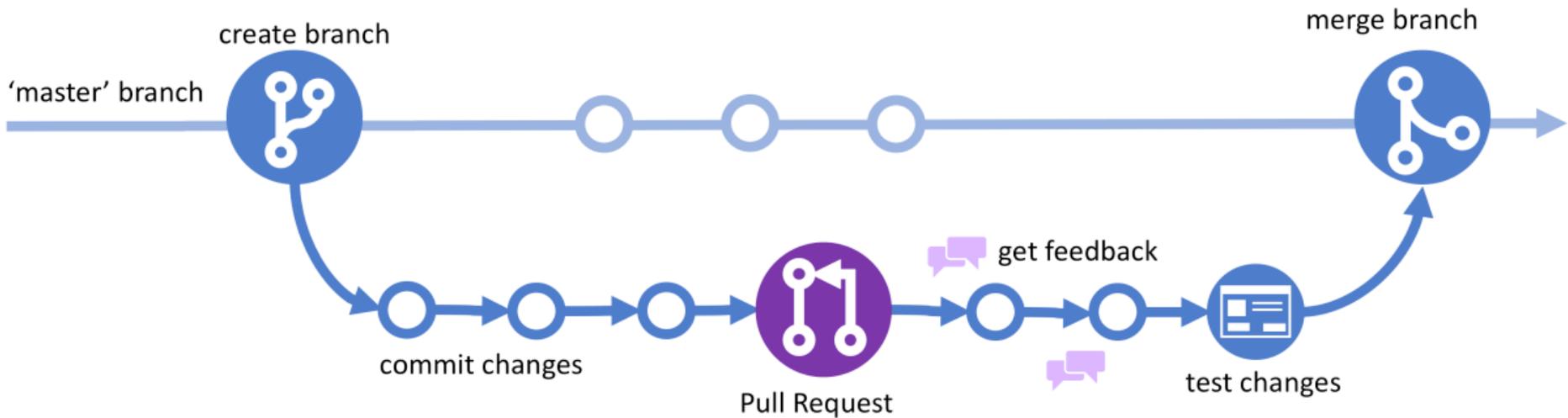
- Versioned libraries, generally consumed at build time
  - Things like apache-commons, Selenified, Spring Boot
  - Semantic versioning is critical
- Version-aware Applications/APIs
  - Public APIs consumed by one or more third parties
  - Runtime versions important, build time not as critical
- “End of the line” applications
  - Generally web or “forced update” applications
  - Semantic versions are less useful

NOTE: Don't tie your pipeline down to just one approach

# GitFlow



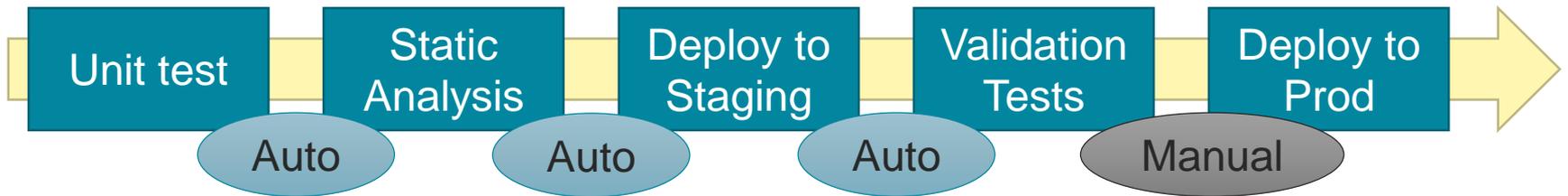
# GitHub Flow



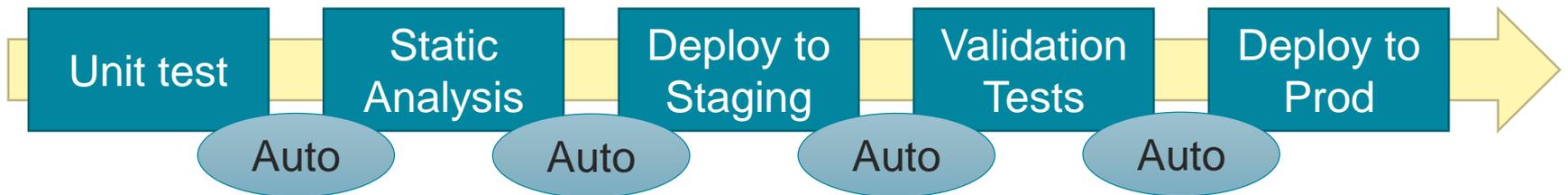


# Continuous Delivery vs Continuous Deployment

Continuous Delivery:

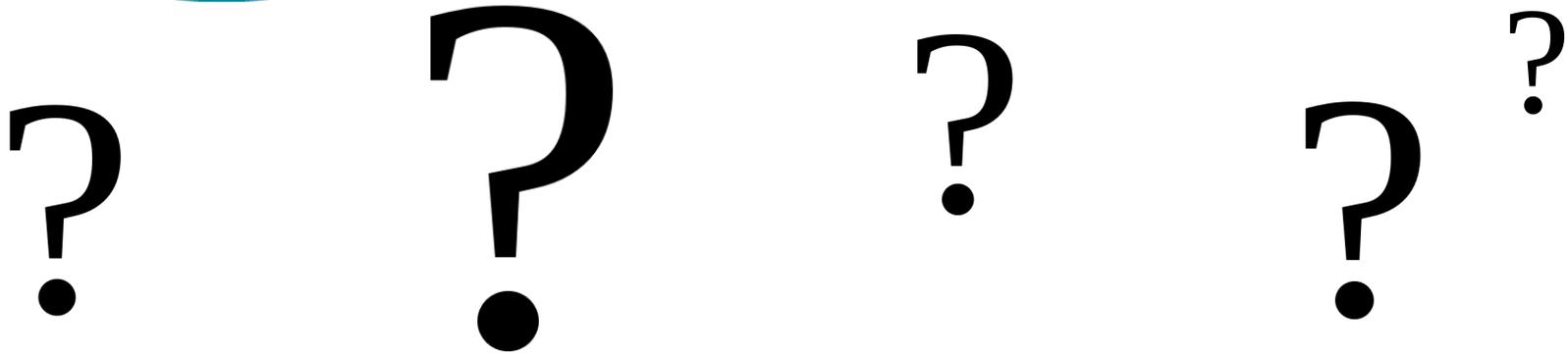


Continuous Deployment:

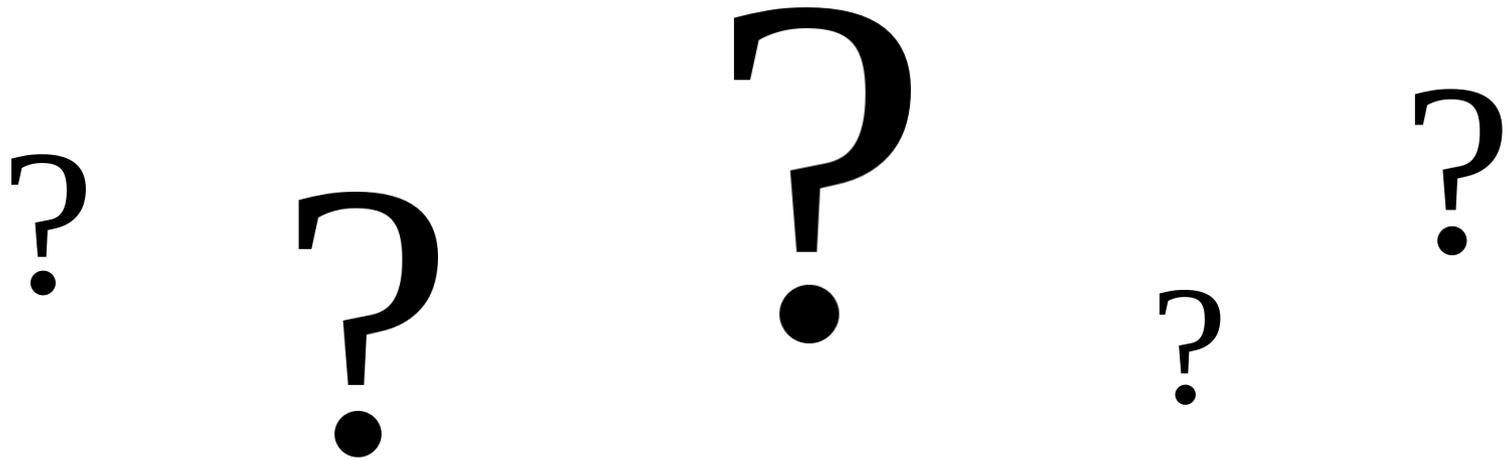




## Question of the Hour



Can and should you **continuously deploy** to **production** for **federal** clients?



# No, BUT

- We can shield the developers with a “developer production”, that gets treated like real production
- Establish **high confidence** early on through **PR Quality Gates**
- Model internal process around deploying to DevProd



# DevOps doesn't just "happen"





# Making it work

Sounds great, but for this to work we need

- Automated testing of *everything*
  - Application
  - Pipeline
  - Platform
- Automated deployments (again, of *everything*)
- Developer buy-in
- Organizational support

# Testing *Everything*

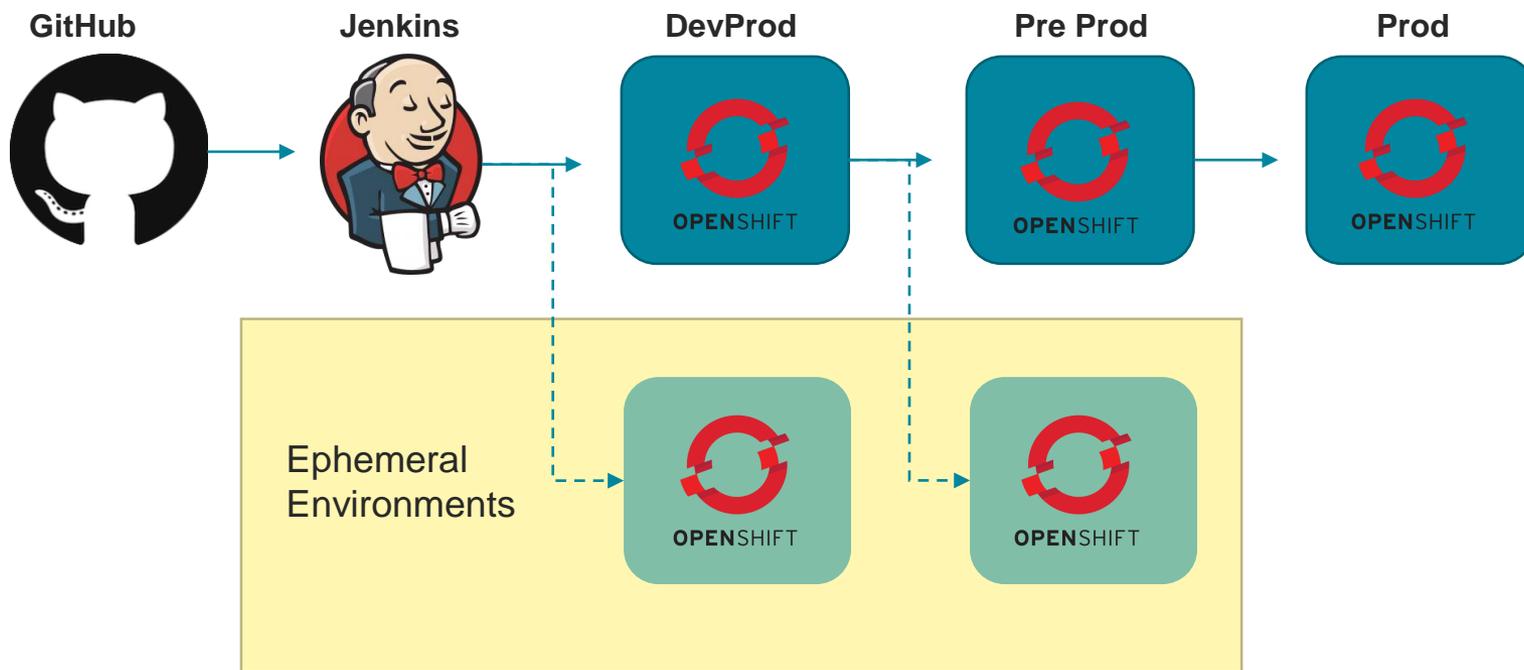


***Cluster redundancy enables platform testing but requires your pipeline be sufficiently generic and parameterized***



# Automating Deployments

In order to do any of this, we need a DevOps pipeline



# Configuration Management

## Pipeline



## Configuration

### Templates:

- microservice-1
- microservice-2
- microservice-3

### Environments:

- DevProd.properties
- Impl.properties
- Prod.properties

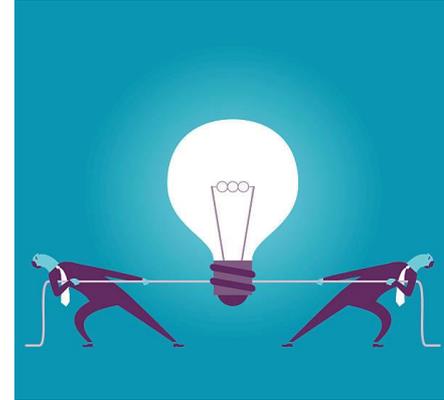
## Cluster





## Fantastic Problems and How to Solve Them

# How things go wrong





# Problem: DevOps In Name Only

*When your developers have no knowledge of or do not care about the pipeline.. but you have a “DevOps” team.*

- **DevOps** reps turn into the **Ops** reps
- Using automation != DevOps

Solution?

- **DevOps** and **Agile** are all about the **cultural change**
- Top-level support and education
- **Replace** those that are **unable to adapt** with those that are



# Problem: Reluctancy to Test

Common symptoms include hearing things like:

- “this is just a prototype”
- “this code is going to radically change later”
- “the UI keeps changing”

Common causes:

- Client pressure to close stories
- Poorly defined definition of done
- Short-sighted management/client



## Solution: Reluctancy to Test



***JUST DO IT.***

(and also)

- Put testing metrics in your definition of done
- Make it easy for testers to write automated tests
- Communication with the client
- Write testable code



# Problem: Back to Silos

*When both intra-team and extra-team communication is overburdened by process.*

- **Low bus factors** are common in **siloed** environments
- Lots of **process** involved in asking a team to **make a change**

Solutions?

- Establish responsibility/ownership, but maintain the “same team” mentality
- Share knowledge within the team
- Leverage horizontal teams to break down barriers across teams



## #Coveros5

1. Everyone needs to buy in to your DevOps process. Management, developers, client, etc.
2. Create a pseudo production environment for developers to use
3. Use on-demand environments instead of static environments for earlier, easier testing
4. Write, run, and maintain tests from the beginning
5. Teach your client about DevOps and they're more likely to cooperate and accommodate your goals



# What do you think?

What are other good ways you can succeed on federal projects?

What's one thing you learned today that you're taking back to your team?

(if we don't finish here, tell me what you think in #devops in the TechWell HUB )

**<http://hub.techwell.com>**