Table of Contents

DevOps for the Agile Enterprise

Background

History is that Scaled Agile has purchased licensing rights to Icon (Mark Rix) material and with be releasing a course based on this shortly. There are a couple of differences in approach but basics are the same. Differences include:

- CALMS (or rather SMALC as the way to present this as Culture comes last) vs SAFe (CALMR) approach
- Materials will change. For example, there is a personal view “helping Mary and Ralph solve their problems” to drive some the exercises.

From introduction:

“In this two-day course, you’ll gain an understanding of the role of DevOps in a Scaled Agile organization. Unlike other training that focuses only on the mechanics of DevOps, Icon’s DevOps for the Agile Enterprise course explores the role of DevOps in the context of the entire IT value stream, and prepares you to successfully plan and implement DevOps capabilities that significantly extend the benefits of Agile transformations. This includes understanding the full- spectrum software delivery ecosystem, where Agile transformations typically are and are not effective; how Agile and DevOps are “better together;” and what people, processes, and technology must be aligned throughout the organization, to achieve true enterprise agility.

Objectives

- Understand prevailing industry definitions of DevOps
- Define DevOps in the context of your enterprise
- Describe the benefits of DevOps and the metrics used to measure them
- Describe the 5 stages of the deployment pipeline
- Understand how DevOps and Agile work together to achieve continuous delivery
- Understand key DevOps roles and responsibilities
- Identify deployment challenges and opportunities in your environment
- Build and prioritize a DevOps transformation backlog
- Launch a DevOps action plan that extends the benefits of Agile”

Attendees

Teacher: Dan James

Support: Adam Beck

Quotes

- “Everyone is doing DevOps; most are doing it poorly.” — Dan James
- “WIP is a surrogate for productivity in the minds of managers” — Unknown
- “Definition of a Defect - evidence of a test that was not written” — Unknown
- “In any value stream, there is always a direction of flow, and there is always one and only one constraint; any improvement not made at that constraint is an illusion” — Eli Goldratt
- “The most expensive words in business: 'we have always done it that way'” — Unknown

Actions

- Good starting point
- Establish DevOps transformation team
- Consider having all SDL (architects) attend DevOps course to help understand framework of thinking (build subject matter expertise)
- Message for FI - “software is eating the world” - what does this mean in our context?
- Publish these notes for others

Materials

- Video “”(Short) History of DevOps“: https://www.youtube.com/watch?v=o7-IuYS0iSE (note: no Phoenix Project)
- Report “State of DevOps” from Puppet (a tool provider): https://puppet.com/resources/whitepaper/state-of-devops-report
- Video “Can You Walk in a Straight Line While Blindfolded”: https://www.npr.org/sections/krulwich/2011/06/01/131050832/a-mystery-why-can-t-we-walk-straight
- Video “One Piece Flow Versus Batch Production”: https://www.youtube.com/watch?v=JoLHKSE8sfU
- Video “Lean - One Piece Flow is Simple”: https://www.youtube.com/watch?v=ciJckWCMvpA
- Video “A Day of Mob Programming”: https://www.youtube.com/watch?v=dVqUcNKVbYg
- Book “Valve Handbook for New Employees”: http://www.valvesoftware.com/company/Valve_Handbook_LowRes.pdf

Key Ideas

- Core idea: DevOps is not about tools or automation. Its about how we deploy work and requires that we understand the deployment pipeline we have today before we start automating.
- The automation paradox “Automation is essential to DevOps but the wrong tools can bury you in a hurry. Don't be hypnotized by shiny objects.”

- Understand that for most DevOps transformations the problem is a fragmented deployment pipeline caused by excessive WIP, cold handoffs, “black box” processing
- Software is eating the world
- And if you don't get out ahead of it you can expect to be part of history
- “We are a technology company with an insurance / banking license”
- Benefits is we can learn faster - software is not static and so can experiment
- Characteristics of a “good” lean implementation:
- Out-learn the competition
- Take learning and out-improve the competition
- Idea of “hypothesis driven development”
- Like the cycle view of the process:
- Build cycle: maximize development productivity through continuous integration.
- Test cycle: maximize delivery efficiency through continuous delivery.
- Release cycle: maximize speed to market though continuous deployment.
- Transition cycle: maximize business value.
- Idea that we should monitor “business value” through measures (eg leading indicators and the business hypothesis) as well as the health of the application.
- Designing “immutable architecture” into our approach.
- Two basic approaches to undertaking an agile transformation (and there is benefit in doing both):
- Top down via value
- Bottom up via stitching things together (i.e. in place create teams, talk to your customrers and improve)
    - Note: DevOps can help drive this approach
- Idea to clean up legacy code:
- Monitor the usage of the code
- Then focus on removing things one-by-one
- This reduces code incrementally
- Result reported: 58% reduction in the number of functions
- Idea on course Kanban
- Set WIP limit to 2 - breaks might mean we are in the middle of an item
- Have volunteer from room move kanban (improve engagement?)
- Have people understand Valve Employee Handbook (we are competing for these people)

Framework

Notes

DevOps Fundamentals

What is DevOps

Base thinking is “agile is good at developing code but this does not reduce time-to-market”. Time to market is the business need. “Agile gets developed; DevOps gets deployed”.

- General positioning is that agile creates, DevOps is about deployment. While this is a hard line (and not really true as any worthwhile agile development worries about deployment and support as well and has to start getting operations involved), its not bad for positioning to help people who are new to DevOps.
- Agile (in the course the context of SAFe) is Concept to Code
- DevOps is Code to Cash

Goal: New code has opportunity to go all the way to release.

Difference between development and operations:

Agile @ Scale (Concept to Code)	**DevOps (Code to Cash)**
Upstream	Downstream
Focus: Design & Build	Focus: Package & Deliver
Analog: New Product Development	Analog: Manufacturing
High uncertainty	Low uncertainty
Change is welcome	Change is **NOT** welcome
Output is non-deterministic / unpredictable	Output is deterministic / predictable
Lean underpinnings	Lean underpinnings

Development and operations mindset start at odds to eachother as development is all about change, and operations is all about stability.

Definition from Wikipedia: https://en.wikipedia.org/wiki/DevOps Note: this has changed from definition given in course:

“DevOps is practice that emphasizes the collaboration and communication of both software developers and other information technology (IT) professionals while automating the process of software delivery and infrastructure changes. It aims at establishing a culture and environment where building, testing, and releasing software can happen rapidly, frequently, and more reliably.”

No one way of doing it.

Why is DevOps Important

Software is eating the world

- And if you don't get out ahead of it you can expect to be part of history
- “We are a technology company with an insurance / banking license”
- Benefits is can learn faster - software is not static and so can experiment

Example: Amazon

- 23,000 new code deploys a day and, at peek times (eg holiday shopping period) this doubles (normal practice would be to lock down / freeze system for fear of breaking something)
- Everything is automated, with exception of coding and peer review
- From development to operations in less than 24 hours (absolute maximum)

Example: Facebook

- You had Facebook Messenger on your phone for 1.5 years and you didn't know it (they were testing it in production)

2016 State of DevOps report details benefits

- 3X fewer defects
- 24X faster recovery
- 200X more frequent development
- 2500X fast time to market

DevOps Principles (CALMS)

Principles:

- Culture
- Automation
- Lean
- Measurement
- Sharing

But actually implemented more as SMLAC (as culture comes last)

Sharing

Start with Sharing

- Need to understand how the process really works (we are all doing DevOps; mostly badly)
- Need to establish reason to do something (out-learn the market, faster time to market)

Break-down silos, visualize how the work is really done today

Form dedicated cross-functional team(s) to address

Know all your customers

Include third parties

Inspect and adapt based on what you are seeing

When you have visualized the work you will get benefits:

- Avoid surprises
- Avoid re-work
- Avoid delays
- Avoid blame-game

Measurement

Then measure so that:

- We can share knowledge about how the process really performing
- We can establish a goal / direction - you have to know where you are going (see Can You Walk in a Straight Line While Blindfolded: https://www.npr.org/sections/krulwich/2011/06/01/131050832/a-mystery-why-can-t-we-walk-straight)

Agree of common (SMART?) objectives / targets

Record baseline

Inspect and adapt

When you measure the work you will get benefits:

- Cuts through opinion and rhetoric and gets to the truth
- Establishes some “fixed points in space” so we can navigate

Lean

Now lean out (streamline) the process

Focus on

- Remove waste (want to get to zero waste, but unlikely)
- System thinking - optimize for the “system of delivery”
- Theory of Constraints
- Littles Law

Batch reduction: “WIP is a surrogate for productivity in the minds of managers” - watch “One Piece Flow Versus Batch Production”: https://www.youtube.com/watch?v=JoLHKSE8sfU

“We want coders to test, and testers to code”

Mature teams must do a few of the XP practices

When you do this you will get:

- Biggest improvements with least amount of effort

Automation

Now automate the process to maximize speed and quality

- Continuous integration / deployment
- Automated testing and provisioning
- Monitoring and telemetry

Don't automate anything until you understand the process and the process is sound

Often lowest hanging fruit is to automate the acceptance testing

When you do this you will

- Be able to deliver value continuously
- Receive immediate feedback
- Maximum responsiveness

Which should lead to competitive advantage

Culture

Now use the outcomes to drive widespread cultural change

Over share the vision and the wins

Focus on

- Publically reward / elevate change leaders
- Re-engineer reward structure and work environment
- Make risk taking safe

The Three Ways

1st Way: Flow

Aim is to get products to market faster by building a deployment pipeline

Metric: Deployment Lead Time

This is why we focus on making work visible, reducing batch sizes and WIP

Build quality in - do not allow defects to flow downstream

- One definition of a defect - “evidence of a test that was not written”

Cost of defect:

- In code: $50 to fix
- In regression: $500 fix
- In production: $5000 fix

This is hard - want it set up so that we have steps 1 to step n always flow in one direction (and never loop backward)

This means you need to measure % complete and accurate at each step to understand where to focus

Theory of constraints quote “In any value stream, there is always a direction of flow, and there is always one and only one constraint; any improvement not made at that constraint is an illusion”

Note: from class you really should not move to the second way until you have the first way done.

2nd Way: Feedback

Goal is to increase value delivered by carrying lessons learned into the next round

Make sure everyone is informed of issues

Business metrics drive this:

- Mean time to resolve (MTTR)
- Business outcomes (which we now monitor in the production system, for example)

This is where we use A/B testing (ie experiment with 2 outcomes and you set the system up, say, so that one group of users see one screen, and another group of users see the other and you use data to understand which meets the business need more effectively)

Inspect and adapt

3rd Way: Continuous learning and experimentation

Goal here is to create competitive advantage and a self-healing organization

We want experimentation and risk taking (safe environment)

Scientific method - “hypothesis driven development”

Self-sabotage as a tool (chaos monkey and its brethren)

Taking many small risks, not one big risk

The Deployment Pipeline

Deployment Pipeline Overview

Quotes from “The DevOps Handbook”

- “The deployment pipeline ensures that all code is checked in to version control is automatically built and tested in a production-like environment.”
- “The role of the deployment pipeline is to ensure that all code and infrastructure are always in a deployable state, and that all code checked in to trunk can be safely deployed into production.”

The Build Cycle

Business objective: maximize development productivity through continuous integration.

Agile context: working software every team increment (team / sprint DoD)

Development responsibility. Aim is to deliver bug free tested code into next cycle. Called “continuous integration” in the slides.

Aim is to be able to do this at least once every 2 weeks. Note: this is lazy - should be able to do on every “story”

Sample / generic build workflow:

- Build: compile source files into deployable binaries and merge local changes with development branch
- Unit test: verify code functions as developer intended.
- Idea here is to enforce “single responsibility” for functions so that unit tests make sense.
- Integrate: merge dev branches to trunk frequently and to verify operational integrity of the code in a production-simulated environment
- May mean that you need to do some work to take a “hairball” and separate into independently deployable chunks.
- Acceptance test: validate stories against acceptance criteria in an integrated, production-simulated environment
- Environment need to be available to the developers and the developers need to use it.

Workflow Step	**Primary Patterns**	**Helper Patterns**	**Measures**	**Trigger**	**Example Tools**
Build	Build automation; Version control	API-driven development; Code reviews	Frequency (how often to build); Cycle time (how long to build)	Code commit	ANT, Maven, MSBuild, Make, …
Unit test	Test automation	Code reviews; TDD	Test coverage; Frequency; Cycle time	Successful build	JUnit, Unit, …

Workflow Step	Primary Patterns	Helper Patterns	Measures	Trigger	Example Tools
Integrate	CI System; Gated (automated) commits	Trunk based development; static code analysis; modular architecture	Frequency; Cycle time	Successful unit tests	Jenkins, Bamboo, VSTS, CircleCI, TeamCity, …
Acceptance test	Test automation	Environment configuration; Service virtualization	Failure rate; Cycle time; Frequency	Successful (CI) integration run	Selenium, FitNesse, VSTS

The Test Cycle

Business objective: maximize delivery efficiency through continuous delivery.

Agile (well SAFe) context: supports effective system demos and, when done right, means you have potentially shippable features every system increment.

General target discussion. You have a certain amount of time it takes you to push a hot fix through, say 48 hours. If this is the case then you should set things up so that all code can run through this at “hot fix” speed, at least initially. Ask yourself “how do you make this happen?”

Sample / generic workflow steps:

- Integration testing: test new features with live connections to connected systems in a production-like environment.
- Regression testing: test existing features of a complete system in a production-like environment. Need to address sunsetting of features as part of this.
- Exploratory testing: manually test sophisticated usage scenarios to discover new test cases. Need to figure out what makes sense here. For example, Amazon only does this type of testing on security issues associated with their payment system. You may also want to consider Session Based testing (see http://www.hanssamios.com/dokuwiki/what_is_session_based_testing) to put a little structure into this.
- Performance testing: provide assurance that system will perform reliably under extreme production conditions. This is where you do all NFR type testing - security, resilience, etc. Are we meeting our SLAs?

Can (should) be all done in parallel.

Workflow Step	Primary Patterns	Helper Patterns	Measures	Trigger
Integration testing	Test automation	Deployment automation; Test data management; Environment configuration automation	Failure rate; Cycle time; Test coverage	Successful acceptance test
Regession testing	Test automation	Deployment automation; Test data management; Environment configuration automation	Failure rate; Cycle time; Test coverage	Successful acceptance test

Workflow Step	Primary Patterns	Helper Patterns	Measures	Trigger
Exploratory testing	Session based testing?	Deployment automation; Test data management; Environment configuration automation	Failure rate; Cycle time; Test coverage	Successful acceptance test
Performance testing	Test automation	Deployment automation; Test data management; Environment configuration automation; NFR's	Failure rate; Test coverage	Successful acceptance test

The Release Cycle

Business objective: maximize speed to market though continuous deployment.

Agile (SAFe) context: release level definition of done. Goal is to be able to release on demand.

Note: you can have a released system that the customer never sees.

This is all about production, so there are no “mocks” in this part of the cycle

Workflow step:

- Configure: rapidly provisioned deployment environments in support of continuous value delivery
- Stage: host fully validated potentially shippable features in a production grade environment, from which they can be released on demand.
- Concept of blue / green system lives here. Have “blue” system. Build “green” system. When you are ready with the green system, switch load (load balancer - one idle, one live) from blue to green system. If things go wrong, go back to blue system. Changes the way we think about the release process. Potentially gives you “zero downtime” capability. See https://docs.cloudfoundry.org/devguide/deploy-apps/blue-green.html for more information.
- Transactional systems are a particular issue for these types of staging approaches. Need excellent queueing system to make successful.
- Deploy: release software changes to production with high frequency and low risk. Note that deployment process needs to be tested as well, in version control, etc
- Deployments don't have to be “all or nothing”. Could deploy based on geography, for example.
- Deployment does not have to make feature available. You could deploy with feature toggle “off” and test that way before turning it on. Other ideas are Amazon “kill switch” and “late binding” (running software decides if code is used)
- Verify: assure deployments behave as expected in production before they are released to end users

Workflow Step	Primary Patterns	Helper Patterns	Measures	Trigger	Example Tools
Configure	Automated provisioning (infrastructure as code); Teams self-service	Version control; Cloud computing; Technical standards	Cycle time (one day or less)	Successful test cycle	Chef, Puppet, Ansible, AWS, Azure, …

Workflow Step	Primary Patterns	Helper Patterns	Measures	Trigger	Example Tools
Stage	Automated environment configuration	Deployment automation; Blue / green deployment; Demo	Cycle time; Frequency (at least bi-weekly)	Successful test cycle	
Deploy	Deployment automation	Automated environment configuration; Self-service	Cycle time; Frequency; Lead time	Customer readiness	Octopus Deploy, MS Release Manager, CA Release Manager, UrbanCode, XLDeploy, etc …
Verify	Test automation; Automated rollback	Selective deployment; Version control	Cycle time (desire 30mins or less including rollback); Failure rate; Rollback cycle time	Production deployment	

The Transition Cycle

Business objective: maximize business value

Agile context: operationalize critical feedback loops. Deployed solutions are viable and have sufficient business value (and we are measuring it)

Workflow steps:

- Monitor: quantitatively measure and respond to system and use behavior in real time.
- Want to monitor the stack but also epic by epic, feature by feature (related to the business case)
- “One week after we deployed this feature, here are the leading indicators so far …”
- Want full stack telemetry with centralized data collection and reporting
- Respond: detect and resolve production issues before they cause business disruption.
- Want proactive detection coupled with rapid fix → release cycle. Set things up so that it goes through “normal” cycle in the pipeline (no shortcuts). In other words, make this cycle fast!
- Need version control so can rollback
- Have tools to track sessions so we can replay and see what happens?
- Immutable architecture is part of this discussion. Only deploy running images. If you need to change an image you need to deploy a new image (see below). Prevents “configuration drift”.
- Stabilize: assure sustainably high levels of business continuity, application service levels, and data protection
- Want no unplanned outages or security breaches
- When you start to get confident with your system figure out tests to try and break it. Eg hack-a-thon theme
- This is where “Chaos Monkey” from Netflix fits in (only streaming vendor that wasn't concerned

when AWS went down as Chaos Monkey had already killed those processes and they knew they were OK).
- There are an army of Monkeys now - Netflix call it their “Simian Army” (see below):
- Deliver: demonstrate tangible business value has been realized and customers are delighted before operations takes custody.
- Learnings here are feedstock into the backlog for the next set of development work
- Want to consistently exceed customer expectation and value (return) targets

Workflow Step	**Primary Patterns**	**Helper Patterns**	**Measures**	**Example Tools**
Monitor	Telemetry	Data collection; visual displays	% of stack monitored; Custom SLAs and KPIs	Dynatrace, App Dynamics, New Relic, Splunk, …
Respond	Proactive detection (work to prevent)	Alerts and notifications; Server reboot avoidance; Cross-team collaboration (original team fixes problems)	Mean time to restore (MTTR)	
Stabilize	Business continuity	Failover / disaster recovery; Cyber security; Design for operations	Outage frequency; Emergency change %	“Simian Army”
Deliver	Business alignment	Product adoption; Transparency; Continuous improvement	(Business) customer satisfaction; (Business) return on investment	

Immutable architecture: From https://thenewstack.io/a-brief-look-at-immutable-infrastructure-and-why-it-is-such-a-quest/

“Immutable simply means something that is created and left unchanged. Immutable things are understood as is, without need to mutate them, and by that have no need to be mutable. Infrastructure of course is word taken from the domain of architecture and physical design of space, for our virtual realms of servers and software applications. Combined we have a created and unchanged set or piece of architecture for hosting applications.

Unchanging infrastructure might sound like the opposite of what you’d want in an agile environment. But what it means is that once the image is working, only a working image is deployed. When it’s time to make changes a new is created, but the previous image is still available for rollback of the environment itself. The fact that we can now create exact, versioned, timestamped versions of an entire environment removes troubleshooting broken instances almost entirely. And thanks to OS-level virtualization, which is spearheading the movement for immutable infrastructure, these images are extremely fast to deploy.”

Netflix's Simian Army (see https://medium.com/netflix-techblog/the-netflix-simian-army-16e57fbab116)

- **Chaos Monkey** disables production instances of systems
- **Latency Monkey** induces artificial delays in our RESTful client-server communication layer to simulate service degradation and measures if upstream services respond appropriately.
- **Conformity Monkey** finds instances that don’t adhere to best-practices and shuts them down.
- **Doctor Monkey** taps into health checks that run on each instance as well as monitors other external signs of health (e.g. CPU load) to detect unhealthy instances.
- **Janitor Monkey** ensures that our cloud environment is running free of clutter and waste. It

searches for unused resources and disposes of them.
- **Security Monkey** is an extension of Conformity Monkey. It finds security violations or vulnerabilities, such as improperly configured AWS security groups, and terminates the offending instances.
- **10–18 Monkey** (short for Localization-Internationalization, or l10n-i18n) detects configuration and run time problems in instances serving customers in multiple geographic regions, using different languages and character sets.
- **Chaos Gorilla** is similar to Chaos Monkey, but simulates an outage of an entire Amazon availability zone.

Kick-starting Your DevOps Program

Define DevOps Objectives

Need to align your objectives and deployment pipeline to the value streams, and you might need to find the these first

Then position DevOps to streamline the Code-to-Cash process

Then analyze the current deployment pipeline and measure:

- Deployment lead time
- Cycle times
- Throughput (units / time)
- Quality (% complete and accurate)

See exercises

Build DevOps COE

DevOps is an enabler at the portfolio level

- Make incremenetal progress at the program and team levels
- Ensure everyone understands the DevOps vision

Like most organizational transformations need to establish a transformation team:

- Guiding coalition for on-going leadership support, providing “air-cover”, budget, impediment removal
- One each from the DevOps deployment pipeline (build, test, release, transition)
- DevOps Transformation Team: Tactical implementation of DevOps
- Core Team (dedicated)
    - DevOps lead practitioners and subject matter experts aimed at practices and tools (build, test, release, transition / operate)
    - Function as an agile team

    - Say 2 people representing each of the main areas in the pipeline (build, test, release, transition)
- Extended Team
    - Advise core team on needed capabilities
    - Typically senior folks from around the organization

Prepare for Action

Value is a function of timely adoption, not raw capabilities. Focus on value. Go all in. Rack up wins. Over-communicate.

Strategies:

- Remember the 3 ways
- Remember CALMS

The automation paradox "Automation is essential to DevOps but the wrong tools can bury you in a hurry. Don't be hypnotized by shiny objects."

Develop the DevOps backlog. For example:

- Epic: Build deployment pipeline that will reduce deployment lead time by 50%
- Feature: Instrument all pipeline activities in a single tool so we can monitor the code to cash cycle
    - Story: Visualize (manual) current workflow
    - Story: Track work items in workflow
    - Story: Measure cycle time for each activity
    - …
- Feature: Add automated testing to the build cycle to reduce rework in the test cycle

Execute and Evolve

PDCA.

Never be satisfied.

Miscellaneous

Version Control

What kinds of things should you put in version control? Everything:

- Requirements
- Tests

- Environments
- Test data
- Experiments
- Configurations
- Procedures (eg check in / out)
- Services
- Binaries
- Rollback capability

Exercises

Assuming you have an idea of your value stream and now want to understand the deployment pipeline assoicated with that value stream:

1. Build the value stream associated with the deployment pipeline as a starting point.
2. Do SWOT analysis on your deployment pipeline.
3. Calculate workflow performance
4. Estimate % Complete and Accurate (%CA) for each step (i.e. % the next step can process as-is)
5. Sum Total Processing Time (Total PT)
6. Sum Total Lead Time (Total LT)
7. Calculate Activity Ratio (Total PT / Total LT)
8. Multiple each step's %CA to obtain Rolling %CA
9. Identify biggest constraint across the entire workflow map
10. Once we have the problem
11. Focus here
12. Do root cause etc
13. Define target state (not end state)
14. Quantify expectation of change
15. Now everyone swarm on that problem
16. Repear
17. Determine KPI's for deployment pipeline. For example:

Cycle	**KPI**	**Baseline Value**	**Target Value**	**Source of Truth**
Build	Build frequency			
Build	Code review %			
Build	Merge to trunk frequency			
Test	% Acceptance tests automated			
Test	% Integration tests automated			
Test	% Regression tests automated			
Release	Environment setup cycle time			
Release	% operational assets in version control			
Release	Deployment lead time			
Release	Mean time to resolve			

What kind of skills do we need?

	Build	**Test**	**Release**	**Transition**
Goal	Continuous integration	Continuous delivery	Continuous deployment	Continuous value
Roles	Developer, Tester, Build engineer, Database developer, Architect	Tester (end to end), Tester (NFR)	Configuration manager, System administrator, Release engineer	Production support, System engineer, Network engineer, Security engineer, Operations engineer
Skills	Trunk-based development, API-driven development, Code review, Environment configuration, Build automation, Continuous integration, Test driven development, Test automation, Service virtualization, Mocking / stubbing, Feature toggles, Application telemetry, Cyber security, Version control	Test automation, End to end testing, Environement configuration, Test data management, Version control	Automated provisioning, Environment configuration, Automated deployment, Selective deployment, Blue / Green deployment, Automated rollback, Version control	Monitoring (full stack), Incident management, Problem management, Disaster recovery

Additional Note

Like the structure of this presentation - practices to “business why”. Some of the mapping might be a little artificial but nice way to hang it all together.

Course, DevOps, SAFe, Learning, Coach, Consultant, Enterprise

~~LINKBACK~~ ~~DISCUSSION~~