An Intro to Developer Operations within $REDACTED Monitoring Solutions

DevOps 101

What is DevOps?

- DevOps is the alignment of people, practices, tools and philosophies to increase an organization's ability to deliver services at higher velocity. This speed enables organizations to better serve their customers, and to compete in a rapidly-evolving market.

What is DevOps?

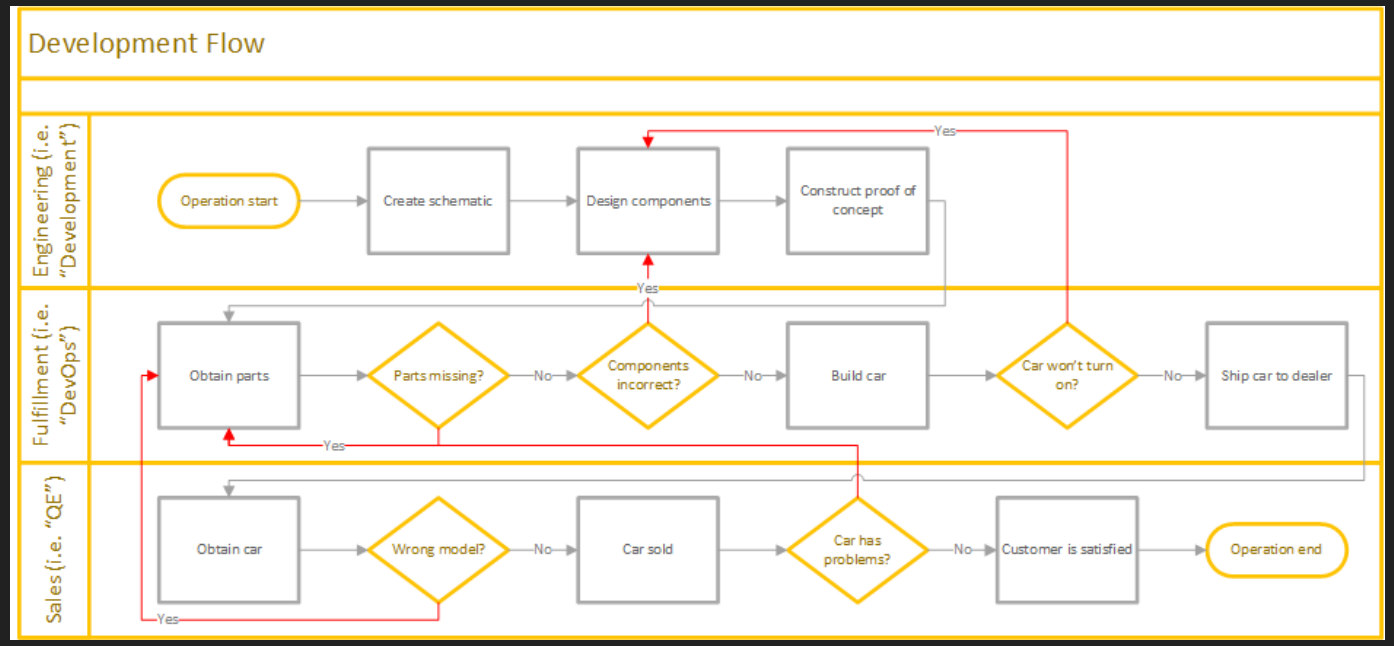
- At the core, DevOps is about the cultural shift towards *Continuous Improvement* (or the philosophy of **Kaizen**, as it's known in Japan).
- The Kaizen philosophy was pioneered by Toyota in the early 1970s, and has since spread throughout the manufacturing industry. Only in recent years have IT organizations begun to adopt this mindset.
- Kaizen's goals are:
  - To ensure maximum quality
  - Eliminate waste within business processes
  - Improve operational efficiency through the standardization of tools and procedures
  - Provide consistent and clear feedback
  - Empower employees to identify areas for improvement and suggest practical solutions
- "If I had an hour to solve a problem, I'd spend 55 minutes thinking about the problem and five minutes thinking about solutions." – (Albert Einstein)

What is DevOps?

- "Improving daily work is even more important than doing daily work." – (Gene Kim, The Phoenix Project)
- Further reading:
  - The Phoenix Project: A Novel about IT, DevOps, and Helping Your Business Win
  - The DevOps Handbook: How to Create World-Class Agility, Reliability, and Security in Technology Organizations

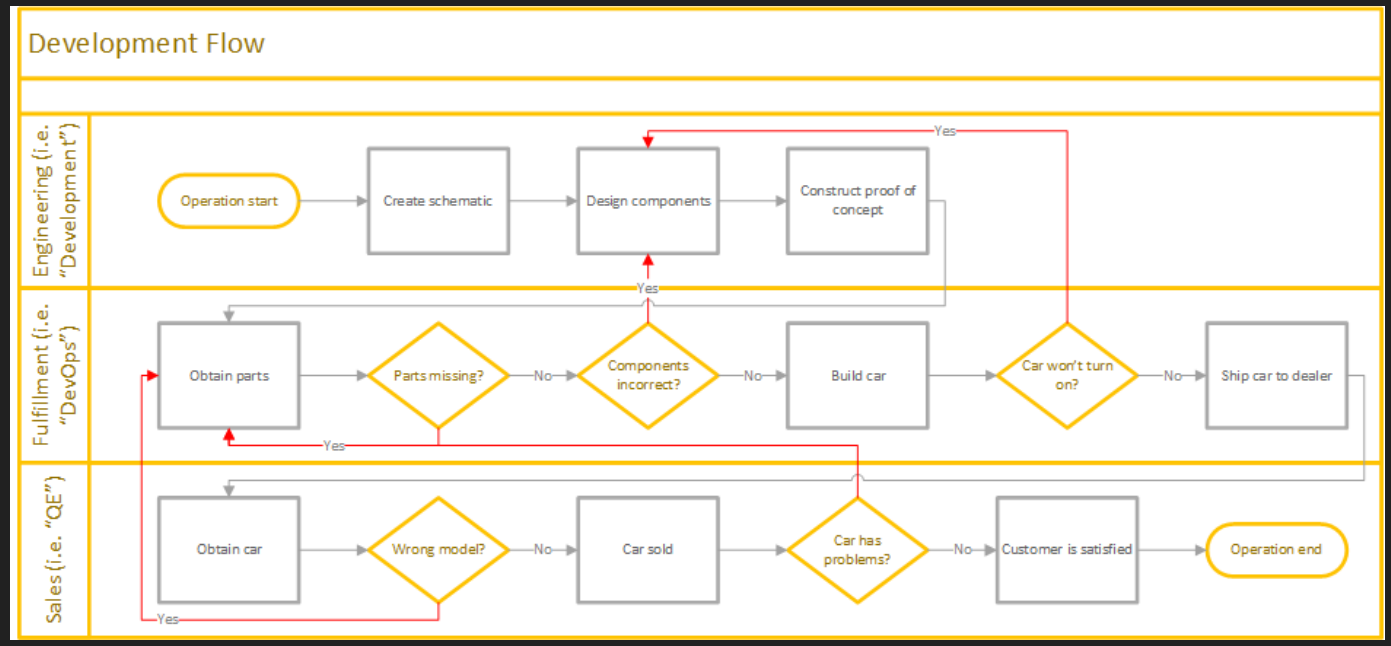
Why do we need DevOps?

- Imagine that you are building a car. Now, imagine that the various teams involved work in total isolation. This assembly line is essentially what we have today:



Why do we need DevOps?

- This type of environment leads to "over-the-wall development" practices.
- OTW practices lead to:
  - Anger & Contention
  - Tears & Frustration
  - Fear & Apathy



Goals

Short-term

- Leverage existing technologies to "buy ourselves time" to work on more important initiatives
- Identify current deficiencies, and discuss paths towards resolution

Medium-term

- Introduce new practices and technologies that make everyone's work easier
- Develop proofs-of-concept that demonstrate the value that DevOps practices have to offer
- Begin to adopt new forms of software development, testing, and management architectures

Goals

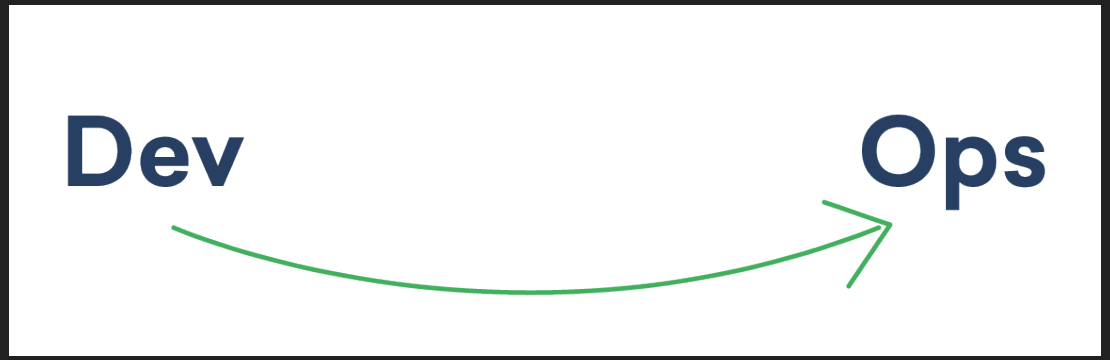
Long-term

- Adopt bleeding-edge technologies and processes that will make us the best DevOps team within $REDACTED Technologies
- Re-architect our software to follow industry best practices (such as the Twelve-Factor App)
- Automate the DevOps team out of work, making every other member of STOP a competent DevOps engineer in the process
- Develop a flexible, modular DevOps migration and knowledge transfer program
- Provide world-class reliability, security and efficiency for our customers

How will we achieve our long-term goals?

The First Way

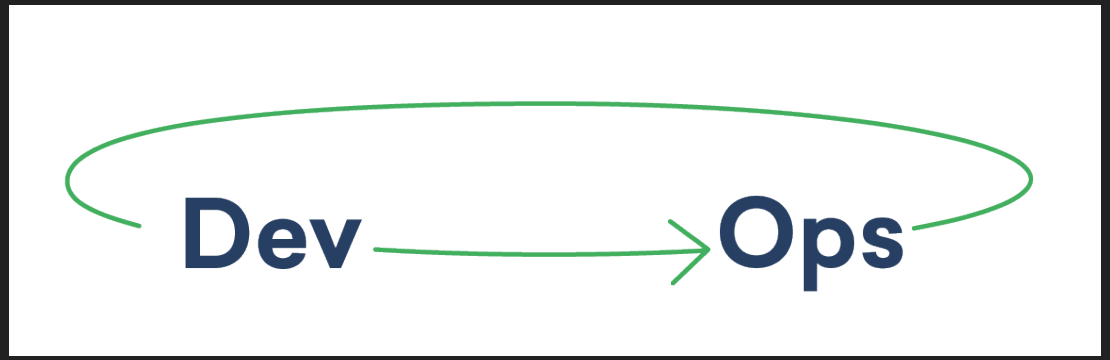
- The First Way states the following about the flow of work:
  - Work should only flow in one direction
  - No known defect should be passed downstream
  - Always seek to increase the flow
- The First Way helps us think of IT as a value stream. Think of a manufacturing line, where each work center adds a component – and thus, value – to the line. Since each work center adds value, it is preferred that each work center does their part correctly the first time around.



How will we achieve our long-term goals?

The Second Way

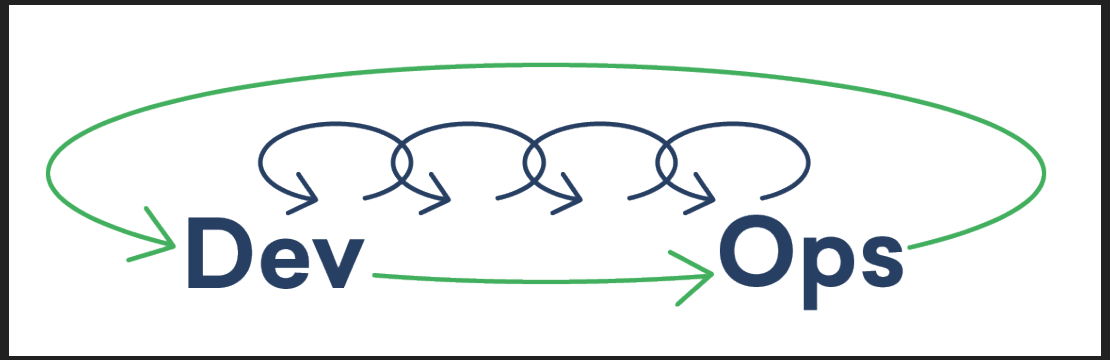
- The Second Way describes the feedback process as the following:
  - Establish an upstream feedback loop
  - Shorten the feedback loop
  - Amplify the feedback loop
- The Second Way teaches us to think of information as a value-addition. When used correctly, feedback can help to optimize the value stream.
  - "Why was there so much wait time at this work center? Resource A was held up."
  - "Why did this process have to be redone? Because it wasn't done right the first time."



How will we achieve our long-term goals?

The Third Way

- The Third Way describes environment and culture through the following practices:
  - Promote experimentation
  - Learn from success and failure
  - Constant improvement
  - Seek to achieve mastery through practice
- The Third Way teaches us that culture and environment are just as important as the work being done. It advocates a culture of experimentation and constant improvement. This results in measured risks and being rewarded for good results.



How will we achieve our long-term goals?

Important Concepts

- Continuous Integration (i.e. Pipelines)
- Continuous Deployment
- Continuous Delivery
- Configuration Management
- Containers
- Environments
- Infrastructure-as-code
- Immutability
- Imperative vs. Declarative Syntax
- Semantic Versioning
- Version Control

Concepts: Version Control

Definition

- Version control is a system that records changes to a file or set of files over time so that you can recall specific versions later.

Git

- Clone
- Remotes (upstream)
- Commits
- Branches
- Merges
- Tags
- Push
- Pull

Concepts: Environments

Definition

- An environment is the computer system (or systems) in which a computer program or software component is deployed and executed. In simple cases, such as developing and immediately executing a program on the same machine, there may be a single environment, but in industrial use the development environment (where changes are originally made) and production environment (what end users use) are separated; often with several stages in between. This structured release management process allows phased deployment (rollout), testing, and rollback in case of problems.

Examples (related to git)

- Development
  - Code committed to the “development” branch
- Alpha
  - Code committed to the “master” branch
- Integration/Reseller/Production/Other
  - Code committed to the “master” branch, and tagged with a version

Concepts: Declarative vs. Imperative

Imperative Syntax

- Code that describes "how" a program should function.
- Example:
  - if $APPLE is "ripe" AND $USER is "hungry"
  - then
    - cut apple
    - eat apple
    - dispose of core
  - done

Declarative Syntax

- Code that describes "what" you need a program to do.
- Example:
  - Eat the apple

Concepts: Declarative vs. Imperative

Declarative syntaxes lead to better imperative code

- Ultimately, most code is imperative. Declarative syntaxes are a high-level abstraction of the underlying imperative code.
- Declarative code tends to be easier to use, easier to scale, and easier to manage.
- However, it requires a deeper-level of understanding from developers about their code. Despite this, the Twelve-Factor App states that declarative code lowers the barrier-to-entry for new employees.
- The Unix philosophy emphasizes building simple, short, clear, modular, and extensible code that can be easily maintained and repurposed by developers other than its creators. The Unix philosophy favors composability over monolithic design. Declarative syntaxes enable this paradigm.

Declarative-friendly stuff

- Most command-line programs
- Most configuration file formats
- All SQL statements
- Most APIs
- Most data storage formats
- Most DevOps tooling

Concepts: Imperative Syntax

According to operations:

- I don't understand this code!
- Why is this so complicated?
- Why can't we re-use this code?

According to developers:

- I barely understand my own code!
- I can't make it any simpler!
- My code is too complicated to re-use across environments!

Example (Bash)

```
#!/bin/bash
# findstring.sh:
# Find a particular string in the binaries in a specified directory.

directory=/usr/bin/
fstring="Free Software Foundation"  # See which files come from the FSF.

for file in $( find $directory -type f -name '*' | sort )
do
  strings -f $file | grep "$fstring" | sed -e "s%$directory%%"
  #  In the "sed" expression,
  #+ it is necessary to substitute for the normal "/" delimiter
  #+ because "/" happens to be one of the characters filtered out.
  #  Failure to do so gives an error message. (Try it.)
done

exit $?

#  Exercise (easy):
#  ---------------
#  Convert this script to take command-line parameters
#+ for $directory and $fstring.
```

Concepts: Declarative Syntax

According to operations:

- This kind of code feels so familiar!
- This configuration is so easy to use!
- I just want to "do the thing" – I don't care "how!"

According to developers:

- I hate YAML!
- I need flexibility, and freedom!
- I don't want to learn a new format!

Example (YAML)

```
logrotate:
  jobs:
    mysql:
      path:
        - /tmp/var/log/mysql/*.log
      config:
        - weekly
        - missingok
        - rotate 52
        - compress
        - delaycompress
        - notifempty
        - create 640 root adm
        - sharedscripts
```

Concepts: Configuration Management

Definition

- Configuration Management refers to the process of systematically handling changes to a system in a way that maintains integrity over time.
- Automation plays an essential role in configuration management. It is the mechanism by which the server reaches its (declarative) desired state, which was previously defined by using (imperative) provisioning scripts in a tool-specific language.

Saltstack

- Salt in 10 Minutes
- States (imperative)
- Pillars (data/declarative)
- Grains
- The Top File
- Targeting
- Orchestrations

Show and Tell

- Salt interactive commands
- Salt Logrotate management
- VSCode usage, features
- Git branch, commit, tag, push, merge
- Salt environment promotion
- Possible: CI/CD

Questions?

- Are our logrotate policies going to conflict with RPMs? Do we need to make an effort to fix those RPMs?
- Can we clear current logrotate policies in /etc/logrotate.d/*?
- Is our Salt code going to conflict with the Linux team's code? Whatever they're doing is a total black box.
- Should we schedule a regular meeting to introduce new concepts?
- Should we schedule regular training sessions to facilitate knowledge transfer?

Are you woke yet?