An OCIO Digital Literacy Course

DevOps For Product Owners

Stephen Curran, Cloud Compass Computing, Inc.

View Online ● Download the PDF

DevOps For Product Owners

Part 1: The Big Questions

1. Introductions
2. What is DevOps?
3. Why is DevOps?
4. Why does it matter me?

Approach

The course will be completed over two sessions covering a mixture of presentation, lab work and most importantly, discussion. Please, jump in at a time with questions, comments, suggestions, snorts, etc. The goal is the material is presented in *your* context.

There will be a couple of labs that will allow you to say - *I done DevOps*

Logistics...

- Any constraints on time?
- Washrooms
- Food and beverages

Introductions

Who are you?

- Project
- Role
- Digital Services experience?
- Please write your name

Me

- I walk the line - business and technology
- Agile/DevOps Development Leader/Mentor
- DevOps since before it was DevOps
- BC Government Projects ICM, JAG and MOTI - *School Bus* and *Hired Equipment*

Are you/Have you been a:

- key technical player on a project?
- key business player on a project?
- core member of a technology project?
- user providing needs for/testing an app?
- user of custom government apps?
- user Outlook, Word and Excel?
- person that refuses to use a computer?

What is DevOps?

DevOps (a clipped compound of "software DEVelopment" and "information technology OPerationS") is a term used to refer to a set of practices that emphasize the collaboration and communication of both software developers and 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. *

Well, that doesn't help much...

* Wikipedia

Why is DevOps?

Roots - merging Developers and Operations work

- Devs - make the code
  - User Interface (UI/UX)
  - Business Logic/Rules
  - Integrations
  - Database (usually)

- Ops - runs the code
  - Servers
  - Networks
  - Databases

NOTE: DevOps is Development Methodology agnostic, but Agile requires DevOps.

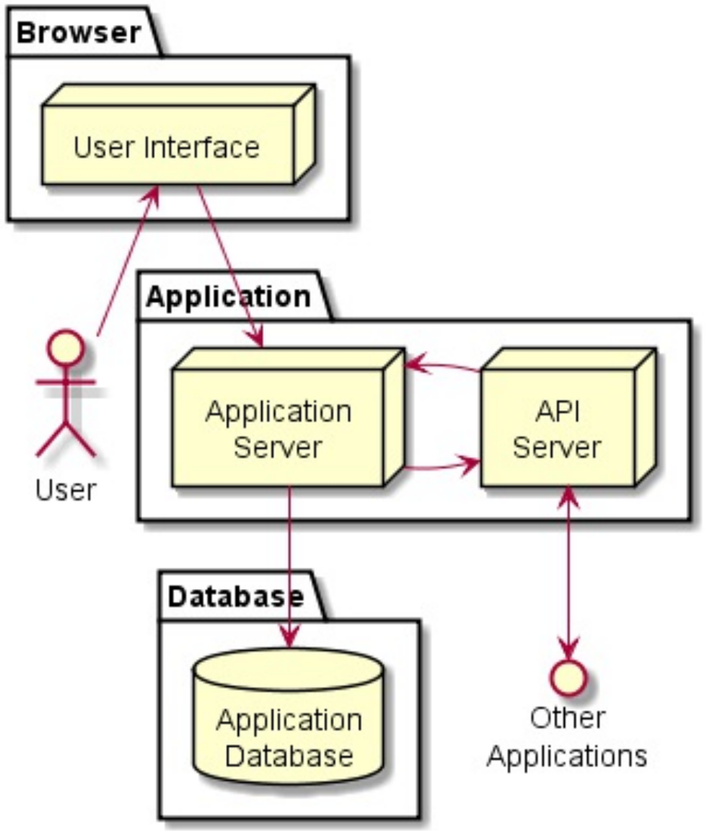
Backup a bit - what's an app?

Examples

- .NET + front end + database
- Java + front end + database
- MEAN (Mongo Express Angular Node)
- Django (Python + front end + database

- Front End: Bootstrap, React, Backbone, Angular, etc.
- Database: Postgres, SQL Server, Oracle, Mongo

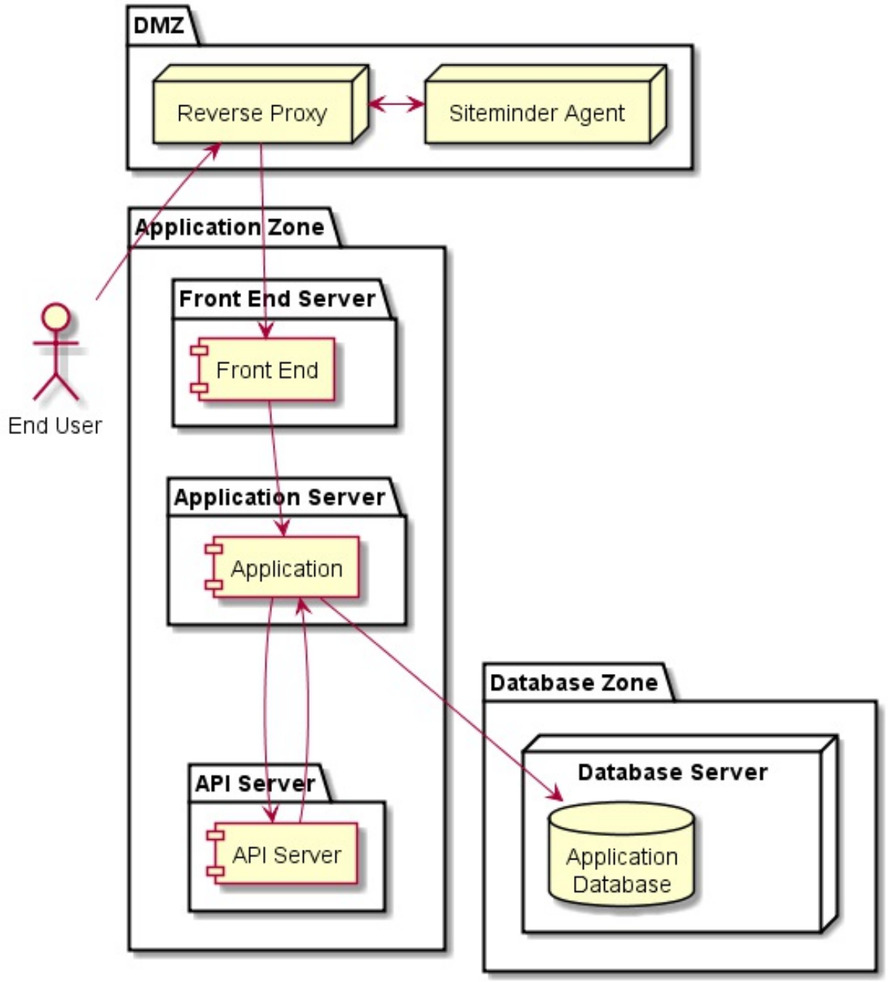
User Stories, usability, logic, rules...



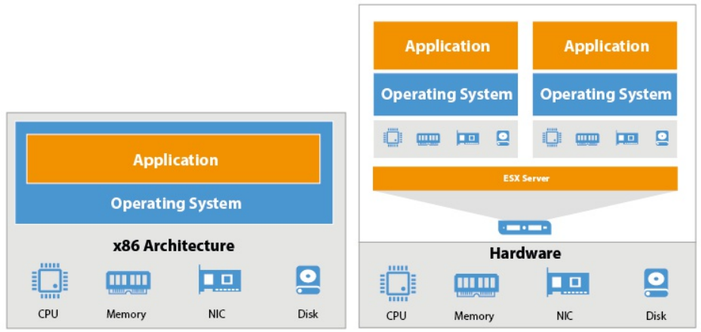
Where does an app run?

Ops View

- Networking zones
- URLs - *https://myapp.gov.bc.ca*
- Authentication - siteminder
- Encryption - SSL
- Firewalls
- Servers
- Storage

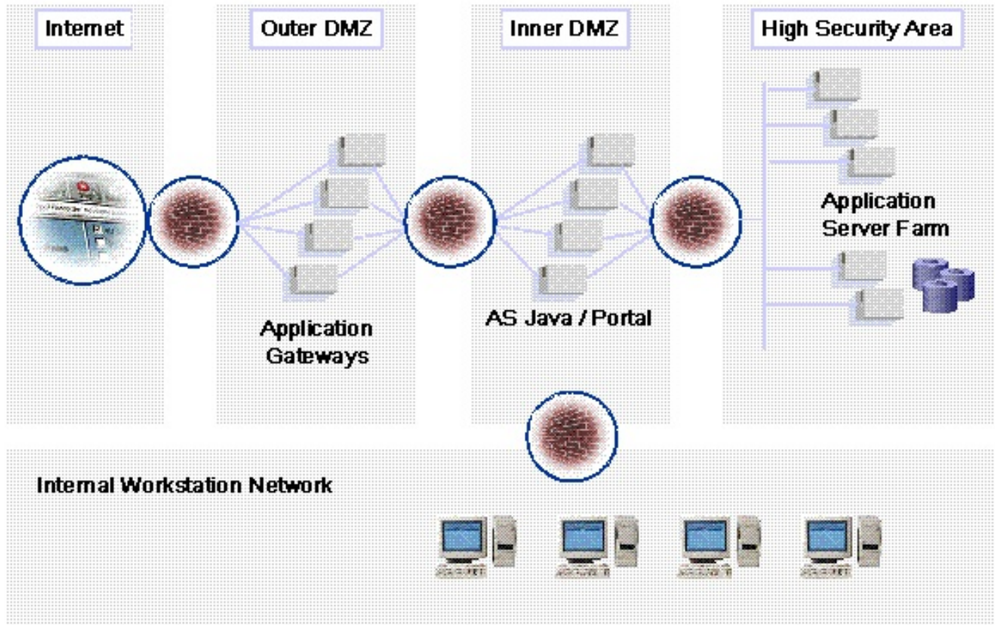


The Building Blocks: Server to Data Centre



Physical vs. Virtual Machines (VMs)

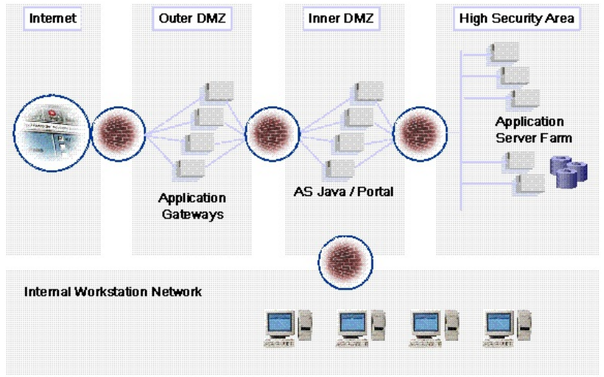
Networking and Security - pre-Cloud

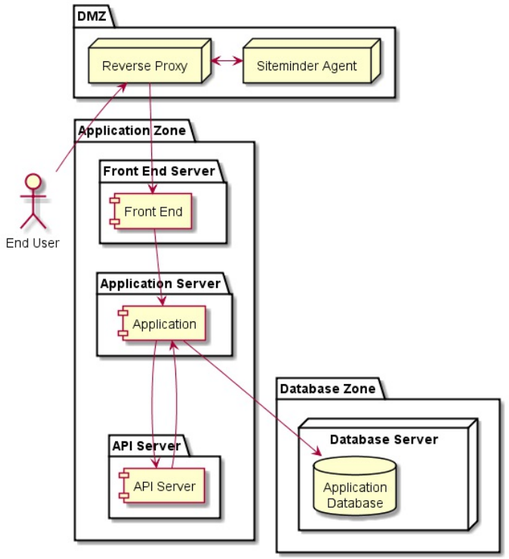


Times 3 - Dev, Test, Prod

Times every Ministry

Our App maps onto the Network





- Configure/secure URL - Siteminder setup
- Procure Compute and Storage
  - Request servers - iStores
    - Physical Servers
    - VMs - Virtual Machines
    - Placement in proper Zone (VLAN)
  - Configure Software servers
  - Install application - when ready
- Establish Connectivity
  - Request firewall updates - iStores
  - Configure software connections
- Changes to compute/network? - iStores

The iStore Impact

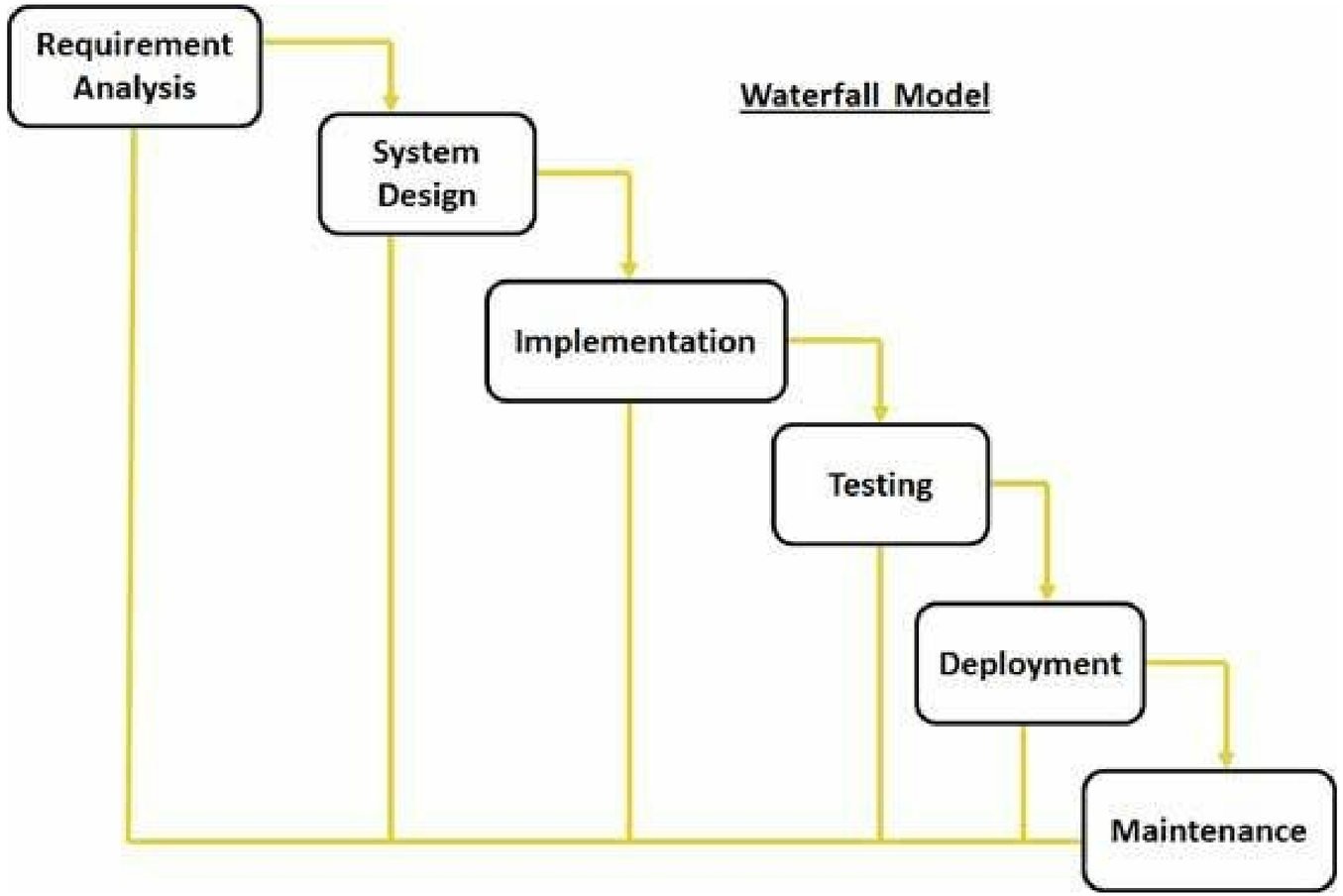
- Tracked - barely
  - Per ministry solutions for tracking assets (beyond billing)

- Money/Time Optimizations
  - Setup environment once
  - Reuse for many small applications
    - Creates dependencies between unrelated apps
    - Forces technology decisions - lock in

- Changes are "difficult"
  - Anything needing an iStore is slow - weeks to months
  - "Innovative" adjustments

- Overprovision big environments
  - Design for the heaviest anticipated load

Dev and Ops: Making it Work - In Theory

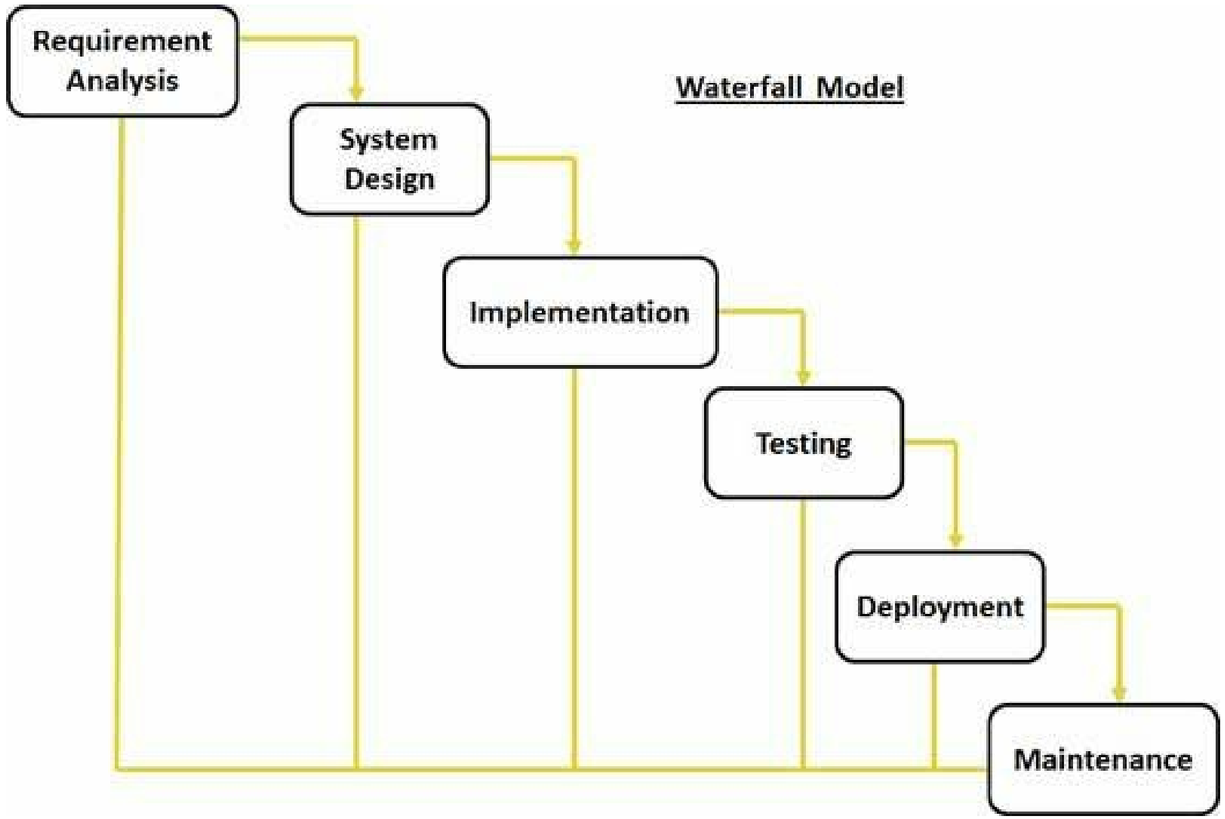


Meetings, documents, agreements and requests

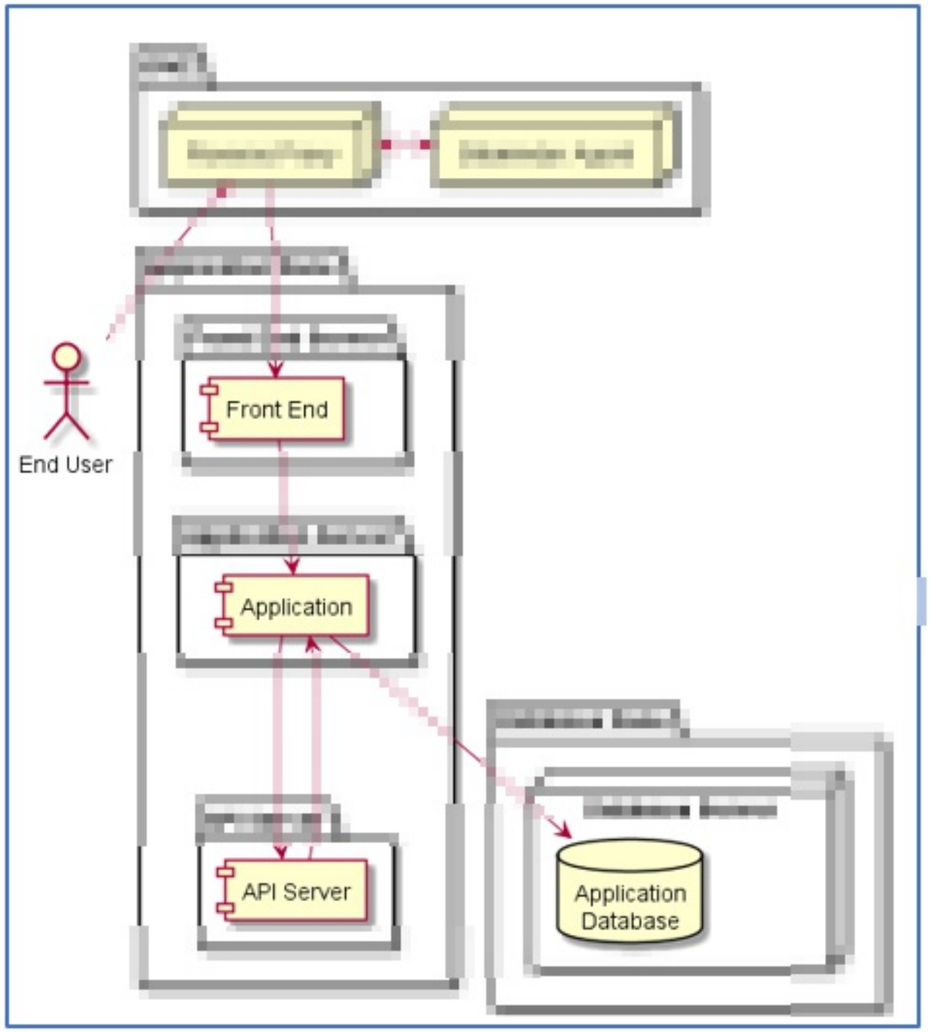
Reality

- Requirements: ***change***
- Implementation: ***takes too long***
  - Devs: Code Development
  - Ops: Procure/Setup Servers
- Testing: ***get skipped***
- As a result, deployment is...

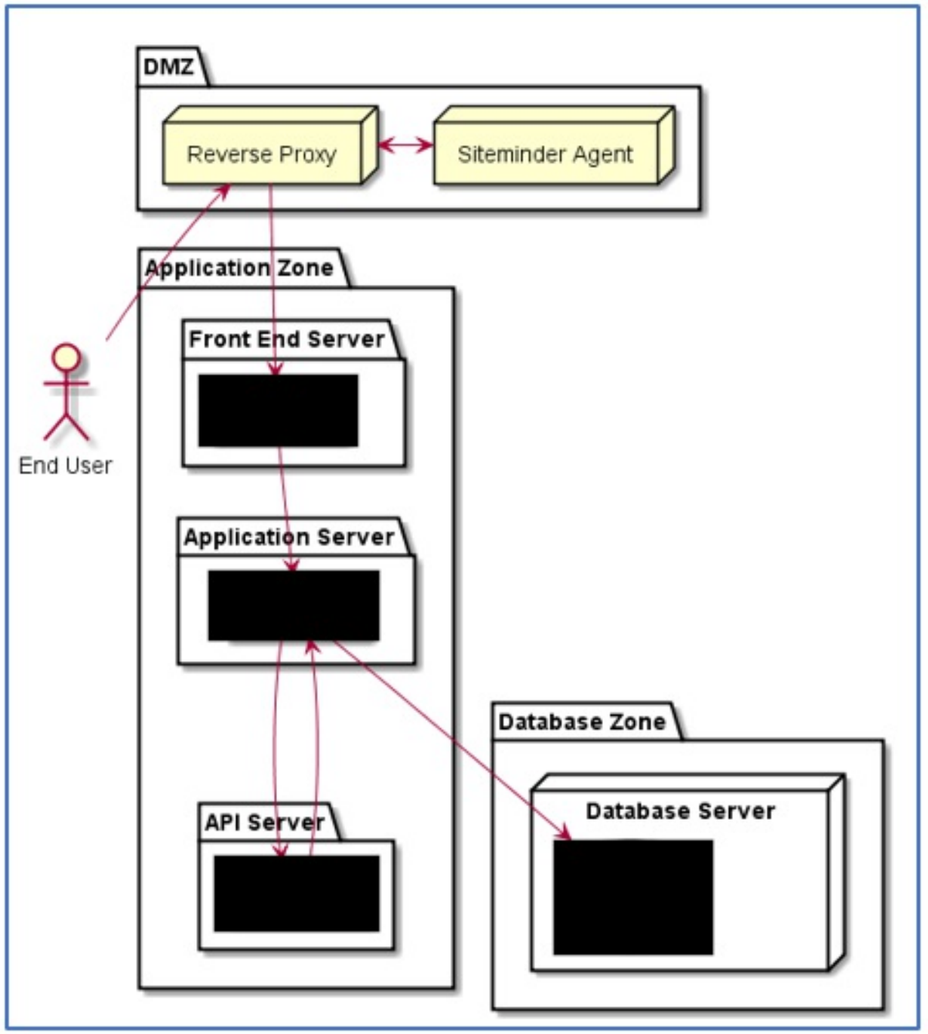
... ***Feared***



What Ginger Hears...



What Developers See

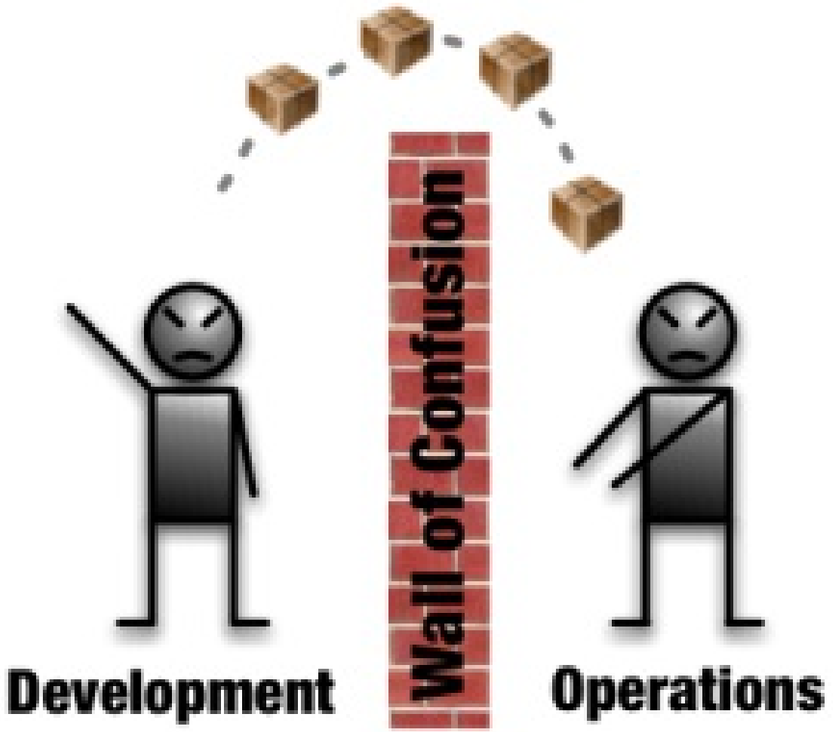


What Ops See

Deployment

The rubber hits the road and...

...up pops *The Wall of Confusion*



What goes wrong?

Ineffective Communications

Communication is via Word documents - the dreaded *Release Guide*

- Premise: To deploy this app, do this...and this...and this...then this...
- Assumption: The writer knows the readers world...impossible

Impact:

- Steps are performed manually
- On-the-fly compensations are made...further invalidating the assumption
  - On Dev, Test and Prod

What goes wrong?

Inconsistent Environments

- Developers develop in their world, deliver to a different one
  - Each Dev creates their own development/test capability
    - Execution environment doesn't match production
    - Minimal test data
  - Periodically delivers code - usually at a milestone - e.g. UAT
    - Agile *should* fix that
  - Test data doesn't match production

Impact:

- It works on my machine!

What goes wrong?

Unnecessary Dependencies

- The *iStore* optimization
  - iStores/funding force optimizations on time and cost
  - Method: Few servers, shared resources

Impact:

- Unwanted dependencies between apps
  - Coordination of multple apps because of shared dependencies
  - Outages of an app because of an upgrade to another

- The Release Party: multiple businesses invovled in one release

All of which leads to...

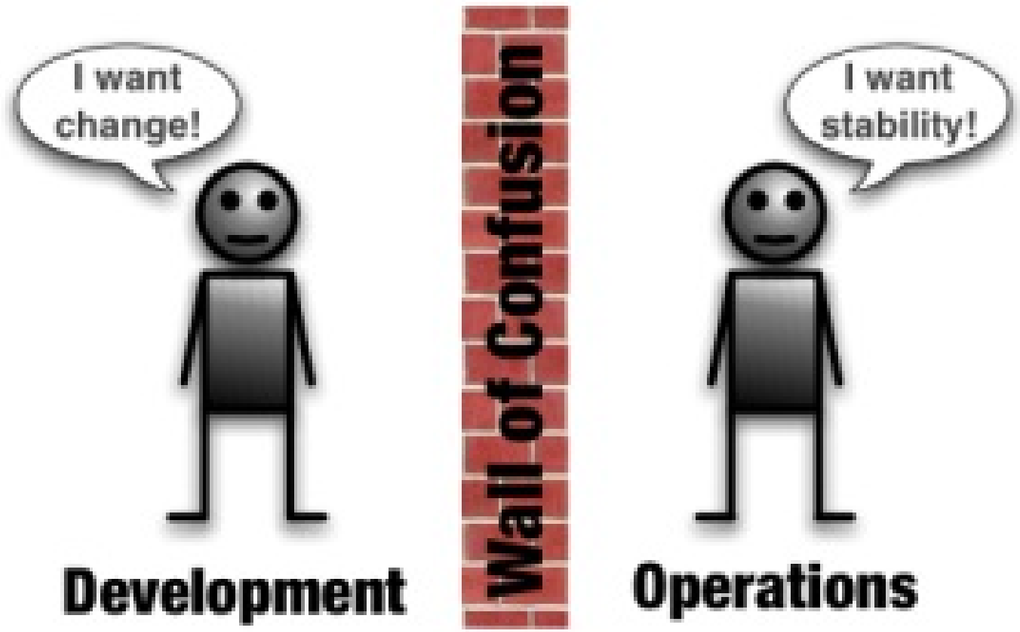
The Day After

CCredit: Twentieth Century Fox - *The Day After*

The Reflex Response

- We are doing it right, we just need to do it *better* next time
- Test more - take longer, check *EVERYTHING*
- For now,though... **DON'T CHANGE ANYTHING!!**

- Except - the users still want more fixes/capabilities



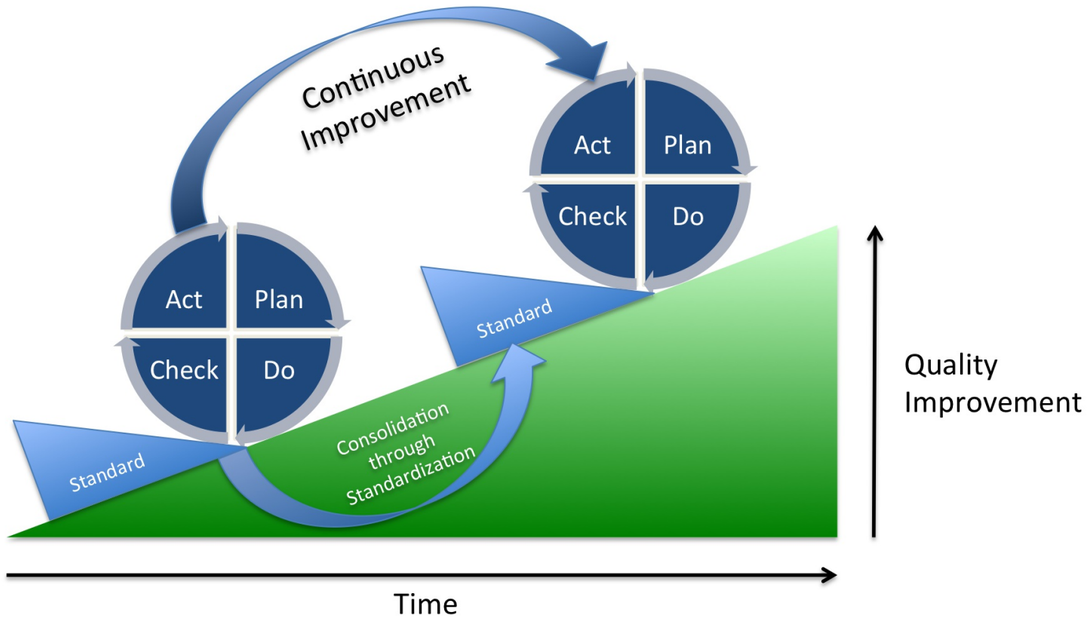
It's a little worse in Government

- Each application is a project - an event
  - Not a product with a lifecycle
  - Focus is on the app, not the long term
- Contracted teams
  - Each starts with own approach & tools
  - Highly variable contact with Ops
  - Improvements are local (team)
- Ops employees
  - Work with many teams...they come and go
- Limited access to data
  - Production-type data
  - Production data volumes

So...What is DevOps?

A culture of applying **Lean principles** to the **end to end** systems:

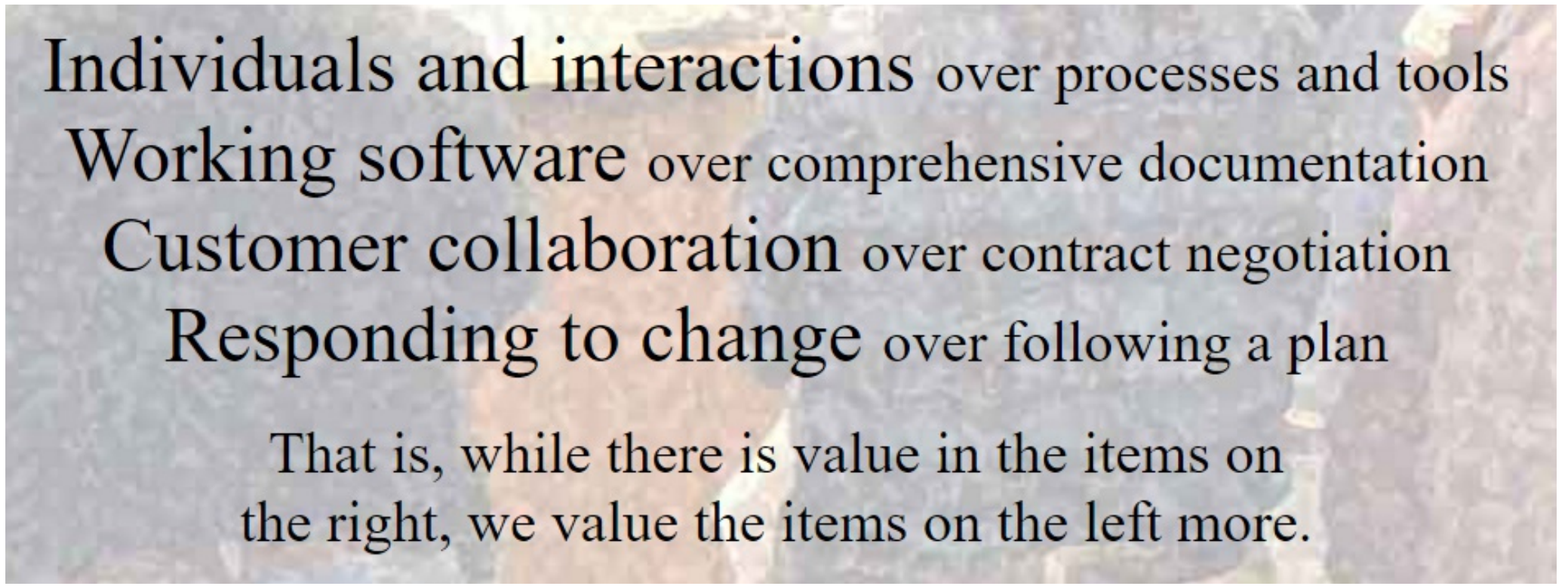
Maximize value; minimize waste



...using some really **powerful tools**

Analogous to Agile

And to some extent, driven by Agile...



* The Agile Manifesto - http://agilemanifesto.org/

Problem: The Release Guide

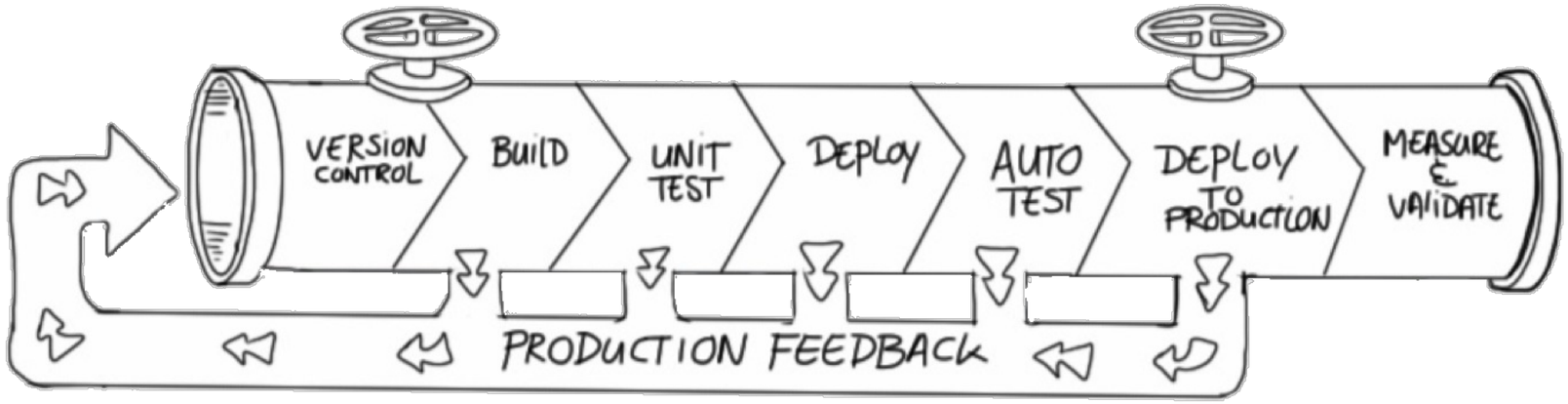
- Old/Miserable: Write It in Word - every step (example: ICM)

Table of Contents

- Better: Write it as a repeatable script
  - Not easy - done incrementally - by lazy programmers
  - ...in isolation

- Even Better: Create and share tools to improve each step

Solution: The Deployment Pipeline



- Subversion, git, github - manage code
- Maven, grunt - build tools
- xUnit - unit test tools
- Selenium, Jmeter - integration test tools
- Migrations, Datical, E-F - database upgrades
- Jenkins - Continuous Integration, job runner

- Connected via triggers

Lab - Deploying an App

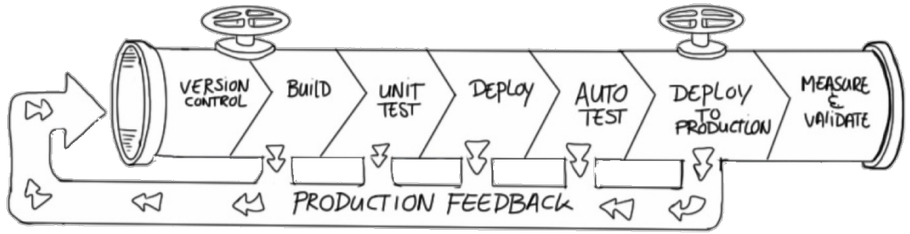
Scenario To Do Web App

- Version control - github
- Architecture - typical Web App

Task Deploy Application

What we learned

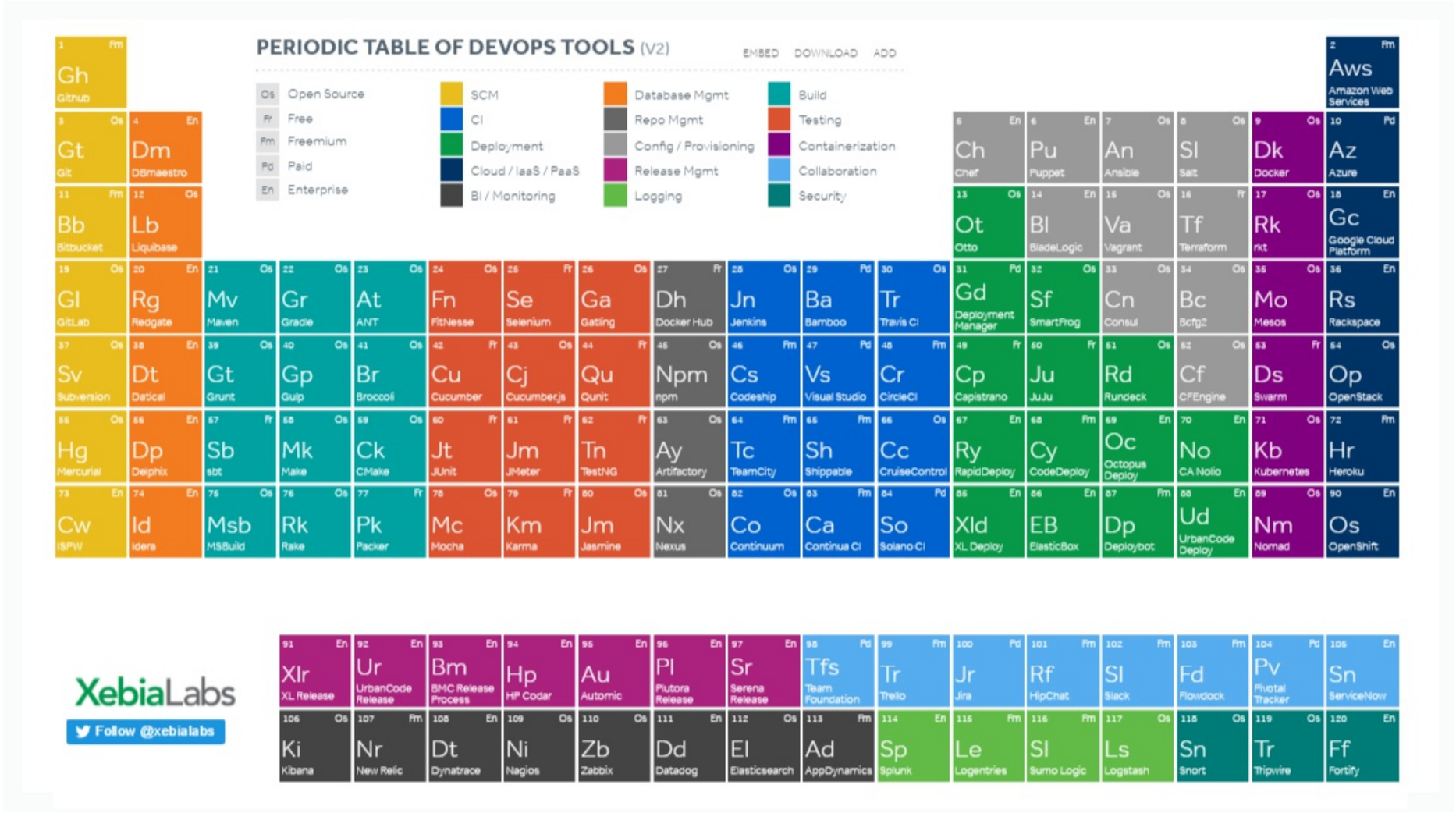
- Automated deployment is possible
- Non-technical people can do it
- There are lots of moving pieces
  - Lots to do if the steps are manual



What we glossed over...

- Setup - Developer and Environments
- Testing
- Promotions - going to Test and Production
- Documentation

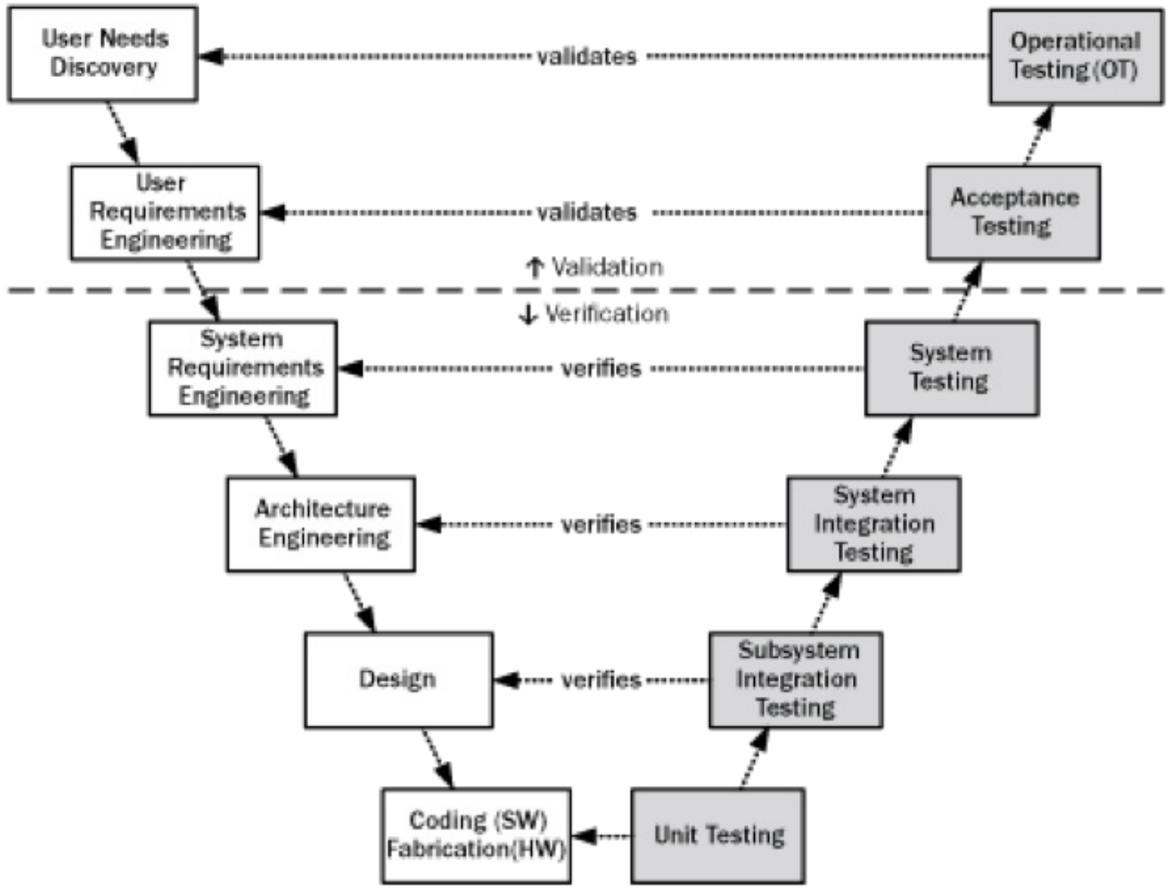
So Many Tools...



Credit - Xebia Labs - https://xebialabs.com/periodic-table-of-devops-tools/

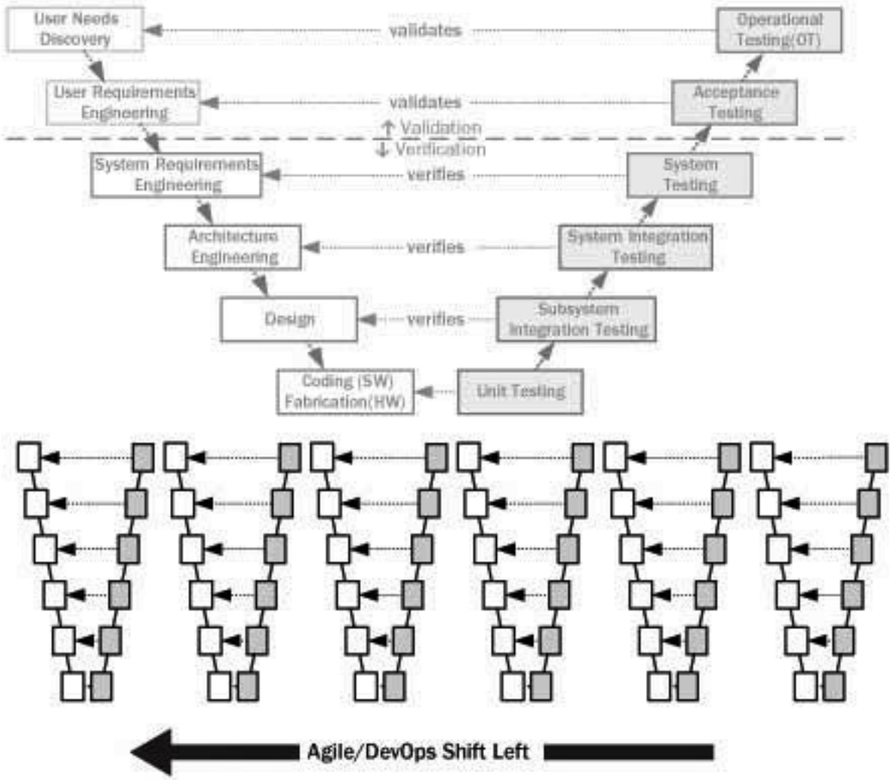
Problem: The Day After

Traditional: Testing V



Problem: The Day After

Solution: Test (and deploy) Early and Often - "*Shift Left*"



Problem: The Day After

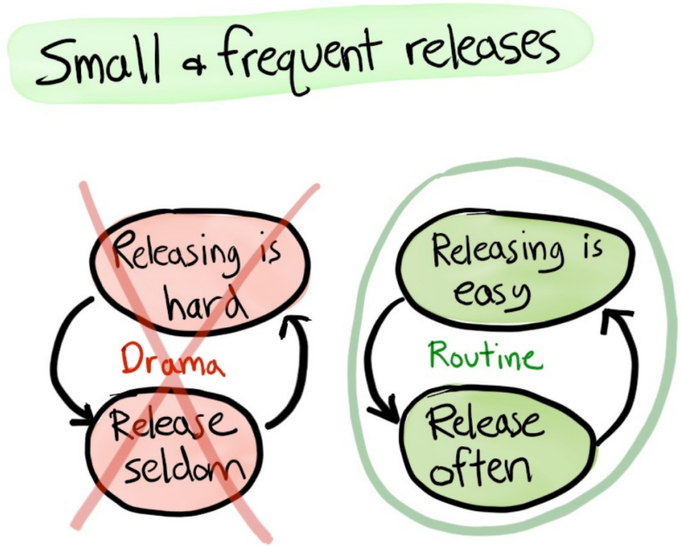
Solution: Really Fast Releases

Done *properly* - aka "Roll-forward"

1. Issue found
2. Issue documented - e.g. JIRA entered
3. Issue investigated
4. Issue fixed, checked in
5. Build / Deploy
6. Verify fix
7. Deploy to Test
8. Verify
9. Deploy To Production...phewwww!!!

Problem: Change is Bad

Solution: Frequent, Small Releases



Big Release - Big Risk - many things to break - hard to fix
Small Release - Small Risk - only a few things to break - easy to fix

Credit: Spotify Engineering Culture - https://labs.spotify.com/2014/03/27/spotify-engineering-culture-part-1/

Problem: Works on my System!

Solution: Consistent Environments

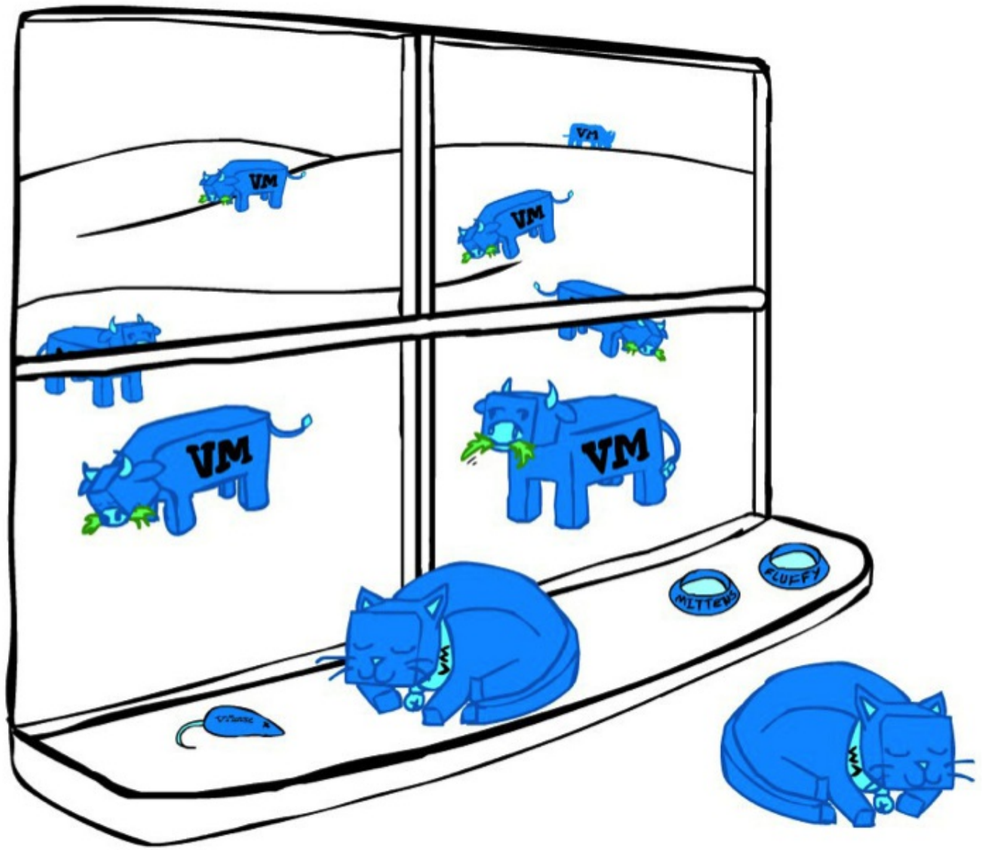
Tools to enable consistency

- Ansible, Puppet, Chef - server setup tools
- Subversion, git, github - configuration as code
- Chaos Monkey - "no pets" verification

Tools so Dev = Production

- Vagrant - VMs
- Docker - Containers

Open source licensing *REALLY* helps here

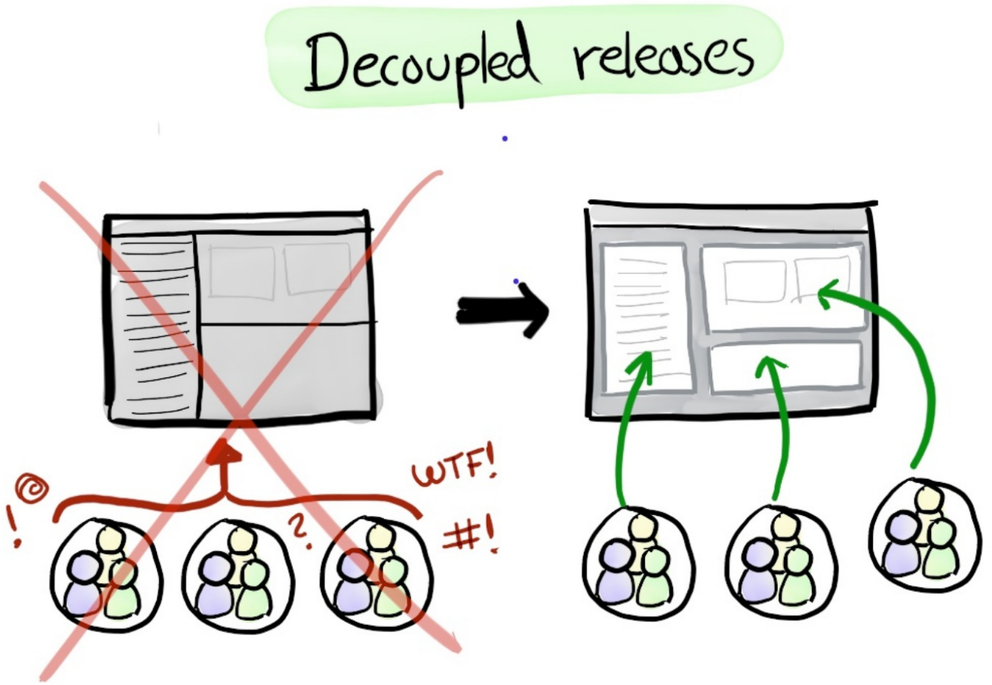


Problem: Unnecessary Dependencies

Solution: Stop it!!

Fake Dependencies

- Enterprise Release Scheduling - don't!!
- Eliminate artificial deadlines
- Dependencies on people
- Dependencies on products/licenses



Architectural Dependencies

- Isolate apps / parts of apps
  - Different servers ($$$)
  - Docker, etc.

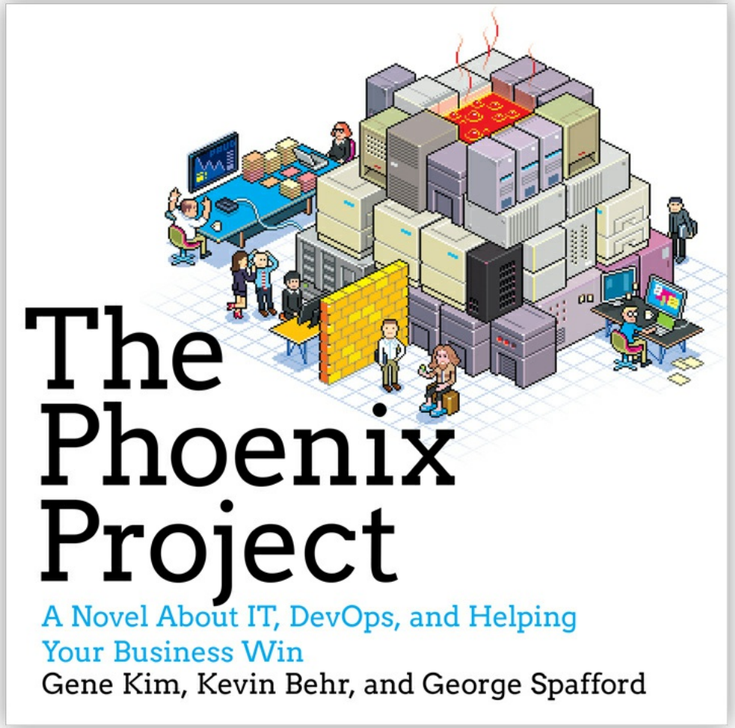
- Don't share databases
  - But don't duplicate data - use APIs

So...what is DevOps?

- A culture of continuous improvement as it relates to the *end to end* delivery of systems
- ...supported by a growing (and standardizing) set of automation tools

The Three Ways

- Systems Thinking
  - Focus on impacts to the *entire* system
- Create Feedback Loops
  - Verify your assumptions/theories
- Continual Experimentation and Learning



Are we doing DevOps??

Anti-Patterns

- Little/no version control
- Devs environments not like Prod
- "Agile" but no DevOps
- Cross Project Release Schedules
- Many manual steps (Release guide)
- Post-deployment fixes without Deploys
- Multi-app Release Party Email Chains
- Server Names - pets (vs. *Services*)
- Test Date = Start UAT Date - 1 Day
- Production Date = Go Live Date - 1 Day

- Day After Syndrome - it hurts!

Patterns

- Version control for everything
- Devs world ~= Prod
- Many, many, many deployments
  - Support Agile
  - Automatic deploy to Dev
  - Triggered deploy to Test, Prod
  - No manual steps - *database*

- Early and often to test, prod
- Automatic feedback and notifications
- App independence

- Day After is just like the Day Before

Why is it important to me?

A **culture** of continuous improvement as it relates to the **end to end** delivery of systems

Everyone's role: fostering the right culture on the team.

It's **your** application.

Starting Place: Transperancy and Visibility

Knowing what's happening on the technical side

- How often are we deploying?
- What is the manual effort?
- How soon can we deploy to Test? to Prod?
  - What is the risk?
  - Is there a way to mitigate that risk?

- How much testing is happening?
- What are the dependencies?
- Are things improving?

Summary

- Architecture
- Developing/Managing Code
  - Version Control
  - Github
  - Open Source
  - Issue Tracking
- Environments
  - Servers
  - Networks
  - Storage
  - Security
- Continuous Integration / Continuous Delivery (CI/CD)
  - Build
  - Test
  - Deploy
  - Verify
  - Monitor
- Visualizing it all

An OCIO Digital Literacy Course
DevOps For Product Owners

Part 2: In The Clouds, On The Ground

Stephen Curran, Cloud Compass Computing, Inc.

View Online ● Download the PDF

DevOps For Product Owners

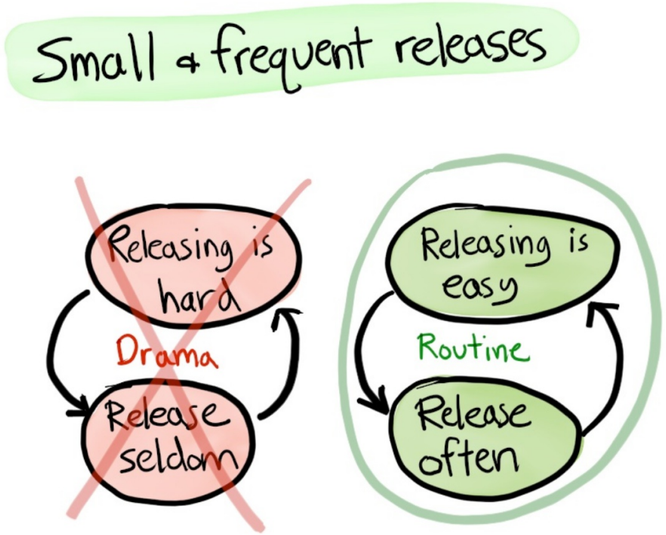
Part 2: In The Clouds, On The Ground

1. Review of Part 1
2. To The Cloud: Models and Options
3. On the Ground: Extending the Pipeline
4. Wrapup: An Application Health Check

Part 1 Review

- Traditional approaches lead to deployment challenges
- Processes reliant on:
  - People
  - Paper
  - Manual Steps

- Evolution historically via increased rigour, fewer releases
  - ...but resulting in higher risks - not lower

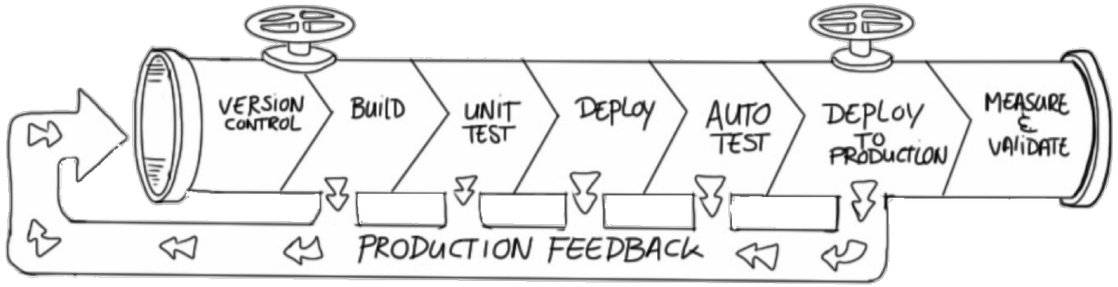


DevOps

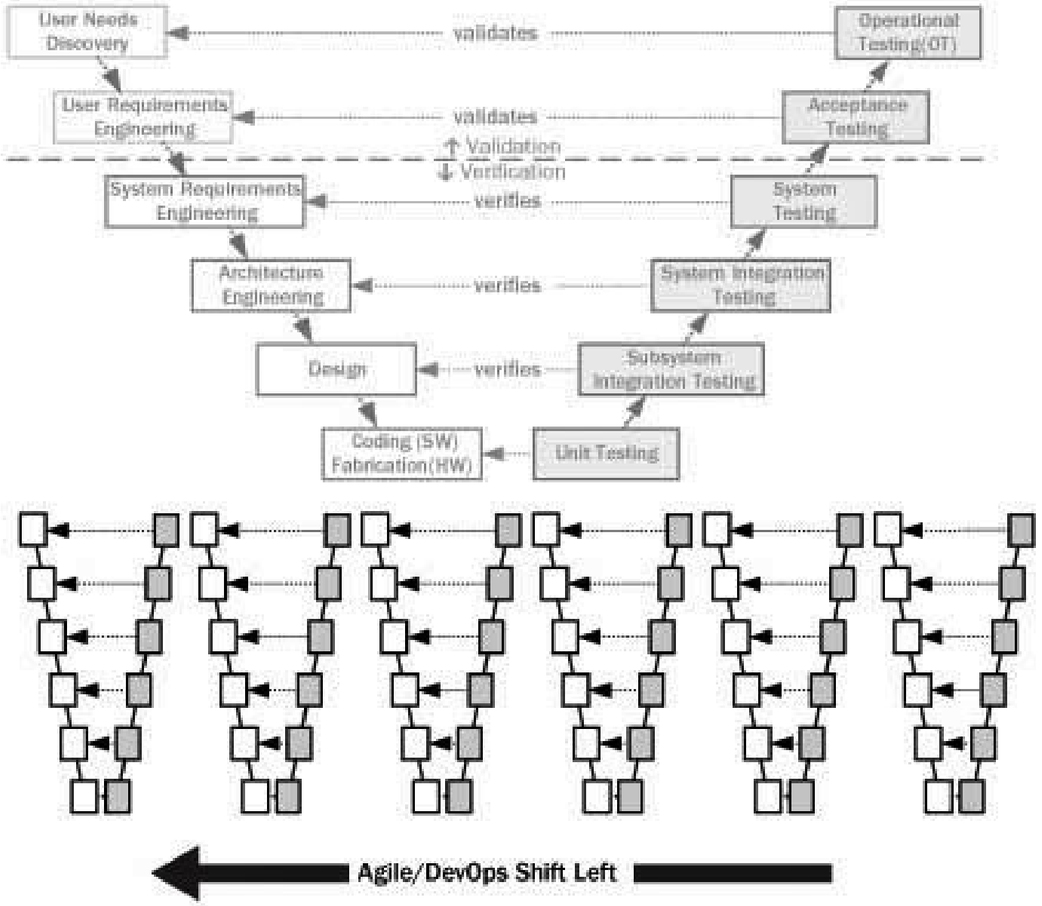
- The application of Lean Principles across the end-to-end system
  - A culture of continuous improvement
  - ...based on a the build up of powerful tools

- Processes reliant on:
  - Standardization
  - Automation
  - Events (triggers) and notifications

- Building up to a deployment pipeline
  - Repeatable
  - Robust



Enabling us to reduce risk by "Shifting Left"



Are we doing DevOps??

Anti-Patterns

- Little/no version control
- Devs environments not like Prod
- "Agile" but no DevOps
- Cross Project Release Schedules
- Many manual steps (Release guide)
- Post-deployment fixes without Deploys
- Multi-app Release Party Email Chains
- Server Names - pets (vs. *Services*)
- Test Date = Start UAT Date - 1 Day
- Production Date = Go Live Date - 1 Day

- Day After Syndrome - it hurts!

Patterns

- Version control for everything
- Devs world ~= Prod
- Many, many, many deployments
  - Support Agile
  - Automatic deploy to Dev
  - Triggered deploy to Test, Prod
  - No manual steps - *database*

- Early and often to test, prod
- Automatic feedback and notifications
- App independence

- Day After is just like the Day Before

What is the Cloud?

- The "Cloud" is the promise of the future
  - Everybody says so
  - What does that mean?

- Often discussed at the same time as DevOps - "Let's just run it in the Cloud"
  - What's the connection?

The Cloud Models: * as a Service

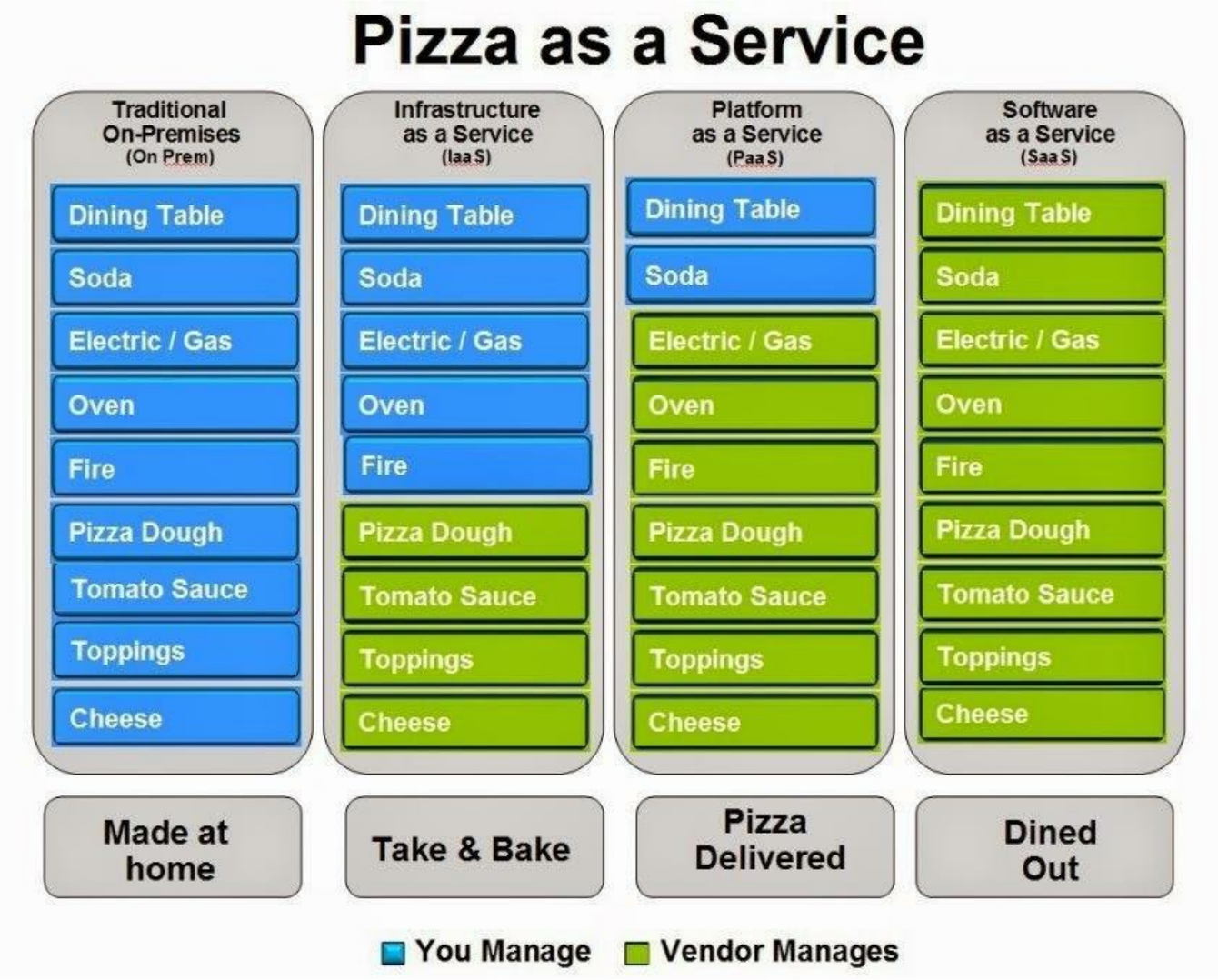
Once you aren't using your computer for computing purposes - you are using the Cloud

- Dropbox / Google Drive - File Storage Service
- Office 365 / Google Docs - Office Editing Service
- Netflix / CraveTV - Video Entertainment Service
- Sharepoint / Shared Drives - Business Storage

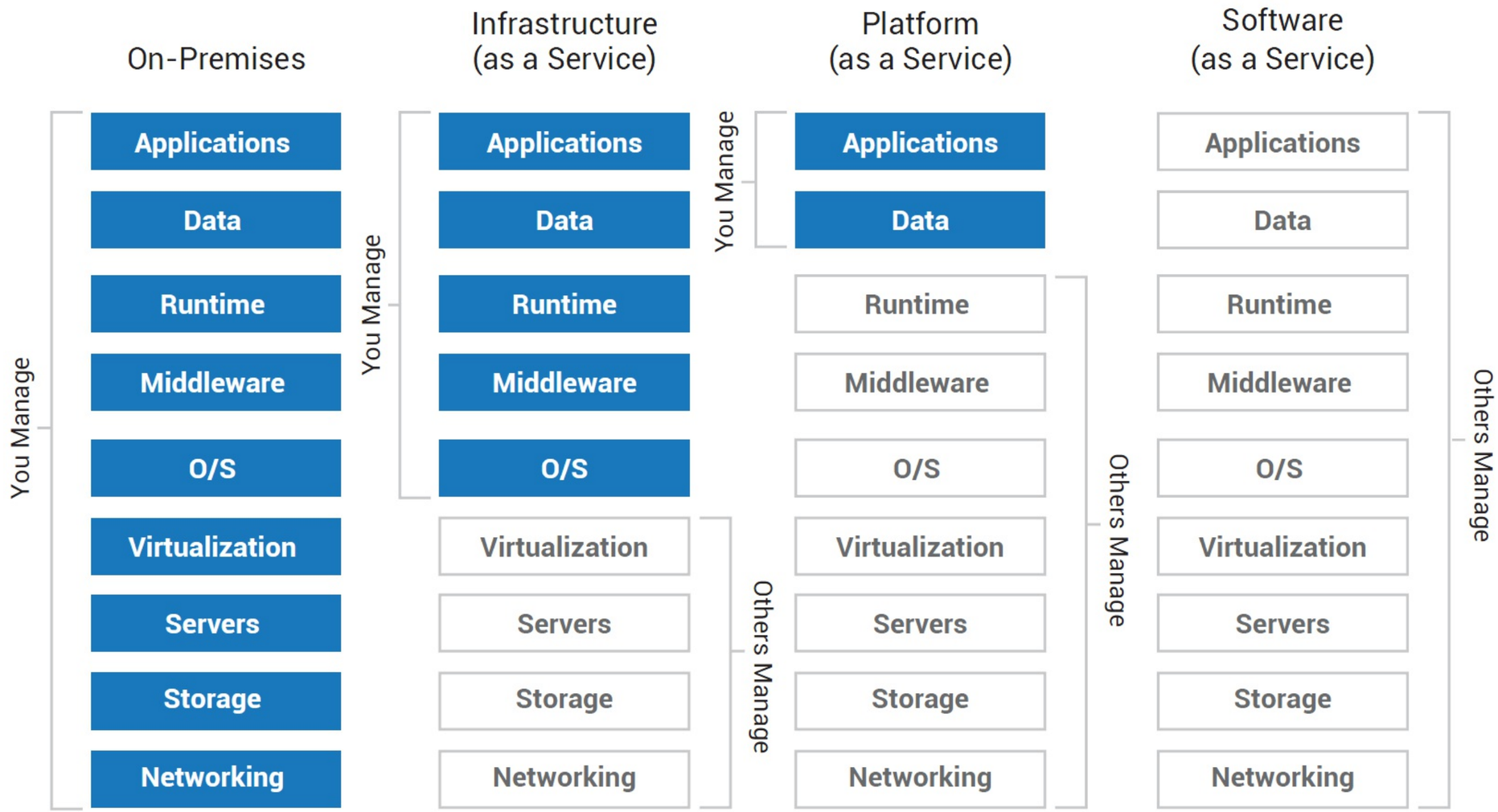
Since all gov't apps aren't on our computers - they are by definition "in the Cloud"

So let's talk * as a Service - IaaS, PaaS, SaaS (but not much about SaaS)

The Cloud Models: * as a Service



On-Premise, IaaS, PaaS, SaaS

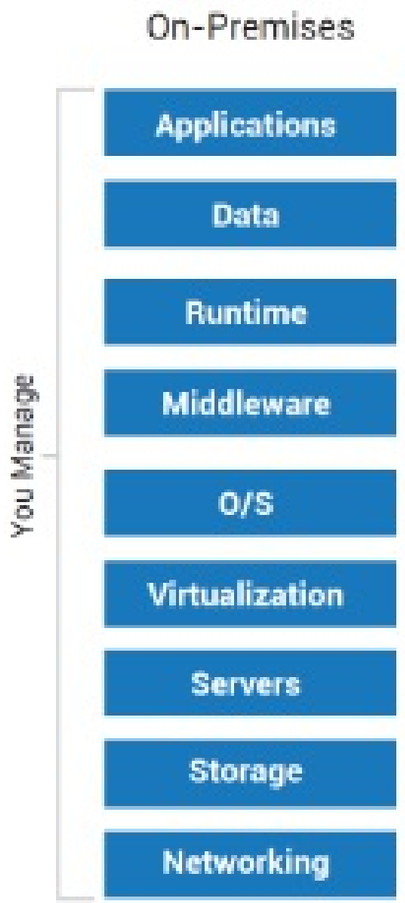


Cloud Service Providers

- HPAS - BC Government Data Centre
- Amazon Web Services (AWS)
- Microsoft Azure
- Salesforce
- Others - IBM, Digital Ocean, etc.

HPAS Services / BC Gov't Data Centres

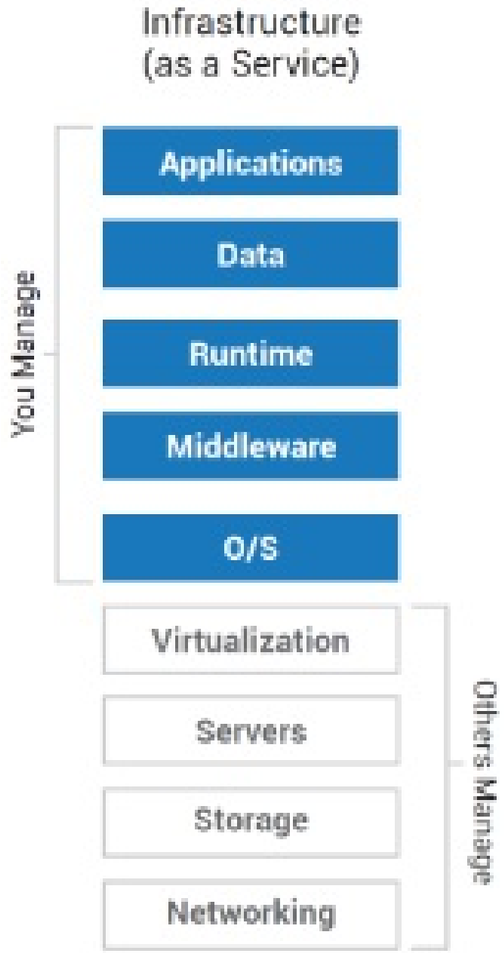
- On-premise-ish, with iStore orders
- Physical and VM servers
- Networking via VLANs and Firewall changes
- Standard images - Windows, Linux
- Standard software - e.g. Oracle, .NET
- No containers (Docker)
- Extra services - patching, monitoring
- Backups and restores
- New! IaaS-ish Offering
- New! PaaS Offering - Red Hat's OpenShift



IaaS * AWS (Amazon), Azure

- Now in Canada
- Procurement: website, credit card
- By use billing - minute and gigabyte
- Spin up servers, configure storage/network
  - Website
  - Configuration script
  - API - programmable - this is **HUGE**

- Resiliency - multiple regions (data centres)
- Hosting expertise - outage responses
- Ground up design to be automated

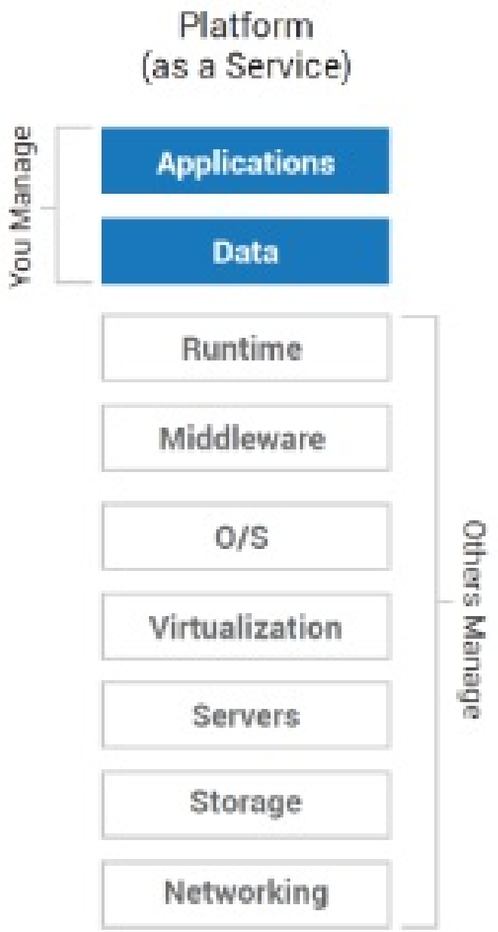


* And a lot more services...

PaaS: HPAS OpenShift, AWS, Azure, Salesforce

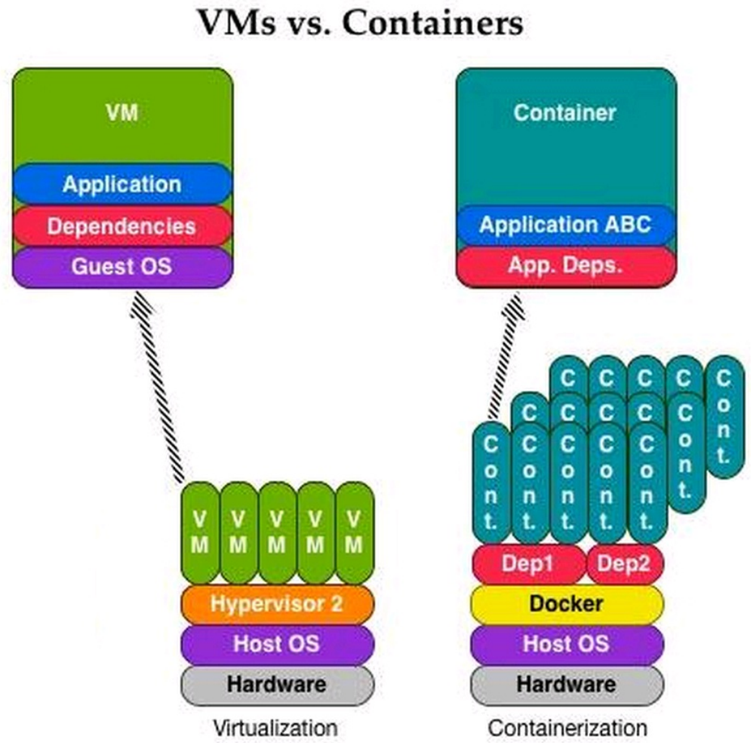
- Declarative needs - concise
- Dynamic setup and configuration
  - Spin up app components
  - Software Defined Network (SDN)
  - True cattle - die (and get replaced)
  - Cattle are less stable - that's Ok

- New technologies
  - Containers (Docker and others)
  - Orchestration (Kubernetes and others)



Digression! Physicals, VMs, Containers

Goals: Resource optimization, consist execution - same everywhere



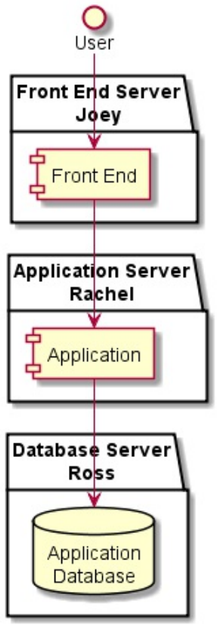
Container Benefits

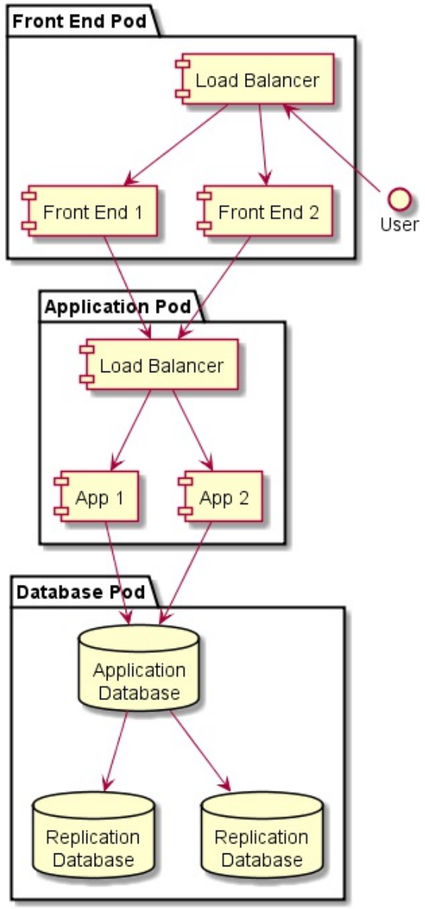
- Easy isolation - *feels like a whole computer*
- Really lightweight - high density - many on one host
- Create once, run everywhere - the "Shipping Container" analogy
  - Run the **same** container on dev machine and Production
  - Drastic reduction in shared dependencies

But...

- New and evolving quickly
- Complex to manage at scale - which is where they are most useful
- True use...as building blocks

Digression! What is Orchestration?

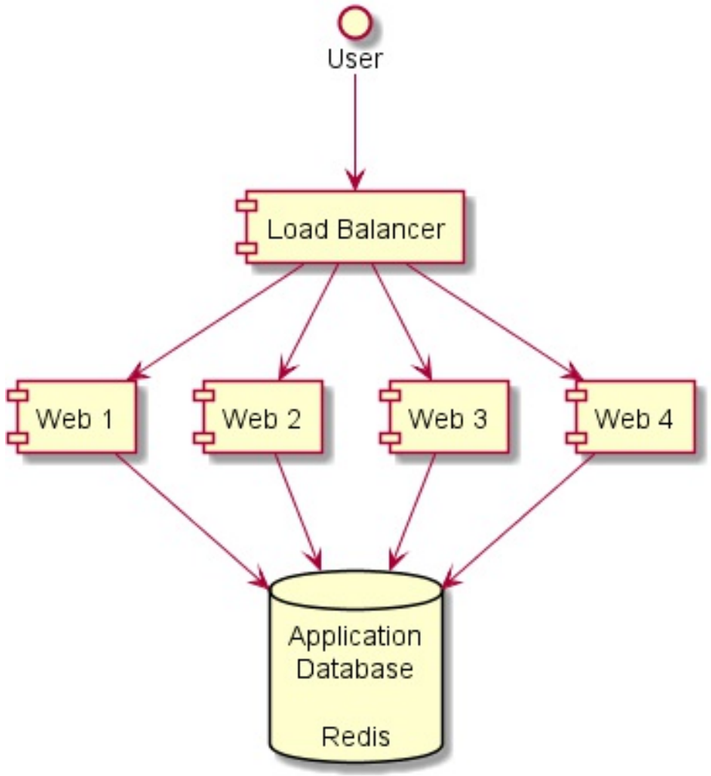




- What if *Front End 1* node crashes?
- What if the load goes up? down?
- What if the main database crashes?
- What if we want to deploy an update?

Orchestration

```
lb:
  image: dockercloud/haproxy
  autorestart: always
  links:
    - web
  ports:
    - "80:80"
web:
  image: dockercloud/quickstart-python
  autorestart: always
  deploy_strategy: rolling
  links:
    - redis
  environment:
    - NAME=Friendly Users
  deployment_strategy: high_availability
  target_num_containers: 4
redis:
  image: redis
  autorestart: always
  deploy_strategy: rolling
  environment:
    - REDIS_PASS=password
```

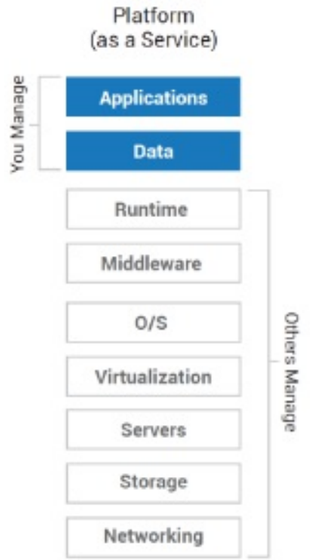


- With all networking connections defined

PaaS: HPAS OpenShift, AWS, Azure, Salesforce

- Declarative needs - concise
- Dynamic setup and configuration
  - Spin up app components
  - Software Defined Network (SDN)
  - True cattle - die (and get replaced)
  - Cattle are less stable - that's Ok

- New technologies
  - Containers (Docker and others)
  - Orchestration (Kubernetes and others)



Devs dream

- Create code
- Declare configuration, evolve it easily
- Deploy easily

Challenges

- New technologies
- New techniques
- For now: Open Source only

SaaS: Office 365, Salesforce CRM

- End users login, use the services
- Some integration with the Enterprise - Single-Sign On (SSO)

Salesforce - also a PaaS

- Roots as SaaS - Sales automation system
- Coming to Canada - running on AWS Canada instance
- Has evolved into a PaaS - **Force.com**
  - Heavy on the "configure" model vs. code - especially the backend
  - Option to build custom frontend talking to Force.com

BC Gov't Salesforce deployments (MTICS, MSDSI, JAG) but have been challenging

- Data Centres in the US - so no personal information
  - Informational apps - no stored data
  - Data Residency handling for personal information

* as a Service Options for Business

Government hosted services are (sort of) easy

- Traditional - with iStores and a DIY-attitude - you are on your way
- IaaS - with a DIY-attitude - you are on your way
- PaaS - with an I-want-to-learn-attitude - you are on your way

Cloud Options are becoming possible:

- Azure, AWS IaaS are in Canada
- Azure and AWS - many other services
- Cloud BC option is coming
- Salesforce will soon be in Canada - on AWS
- Concern is governance - where are all the apps, where is the data?

Cloud Summary and Directions

Four major Cloud models - On-Premise and I, P and S aaS

- HPAS supports (mostly) On-Premise, also IaaS and PaaS (Red Hat OpenShift)
- New public cloud options are coming - but not easily obtained
  - Cloud BC could enable those options
  - Potential: Private/Public Cloud - capabilities extend to use Azure/AWS resources
- Direction is towards PaaS - user expectations will require PaaS/Orchestration
  - Robustness
  - Capacity, Density, Isolation
  - Scalability
  - Ease of Deployment/Management

Break and Lab - The Extended Pipeline

Extending the Pipeline

Demonstrated pipeline

- Deployment
  - Commit a Code Changes
  - Build the Code for deployment
  - Deploy the Code
  - Test the Deploy
  - Promote the Code

What else can we do?

- Shift left!
  - Test and Verify on every deploy
- Monitor and learn

DevOps and *Automated* Testing

- Unit Test - chunks of code
  - TDD - Test Driven Development
    - Write Tests
    - Run the Tests - fail
    - Write the code
    - Run the Tests - until they pass
    - Commit both - the code and test
  - Created during development
  - Run pre-commit - prevent regressions
  - Verification during/post-build
  - Product Owner: Visibility/Transperancy
    - Are we adding unit tests?
    - Are the tests passing?

DevOps and *Automated* Testing

- Integration Test - units in combination
  - Example: app code updates database
  - Created during development
  - Executed post-build
    - To verify the build

  - Can be defined using **BDD**
    - Behaviour-Driven Development
    - Suite of executable user stories

- Product Owner: Visibility/Transperancy

Aside: Manual Testing

- Manual Testing / UAT
  - Combinations of data/unexpected activities
  - Edge conditions - short text, long text, 0s, etc.
  - UAT - Tracked test cases/executions - e.g. Zephyr within JIRA
  - Ideal - automate them as Regression Tests
- Use DevOps
  - Collaboration - when/what to test
- Usability Testing
  - Does it meet the business need?
  - Is it easy to use?
  - Types:
    - Product Owner + Developer - during development
    - Hallway testing
    - Formal Usability Testing

Aside: Formal Usability Testing

- "Typical users" - but not engaged in the project
- Process:
  - Brief Introduction
  - Assign business tasks to be completed
  - Watch, take notes - don't help!

- What is easy?
- Where do they struggle?

- Decisions:
  - Easy to change? Fix
  - Easy to understand after a one-time explanation? Leave
  - Will require training
    - Trade off: cost to fix once vs. cost to train everyone

- Expert Review
- A/B Testing

DevOps and Testing - User Interface Testing

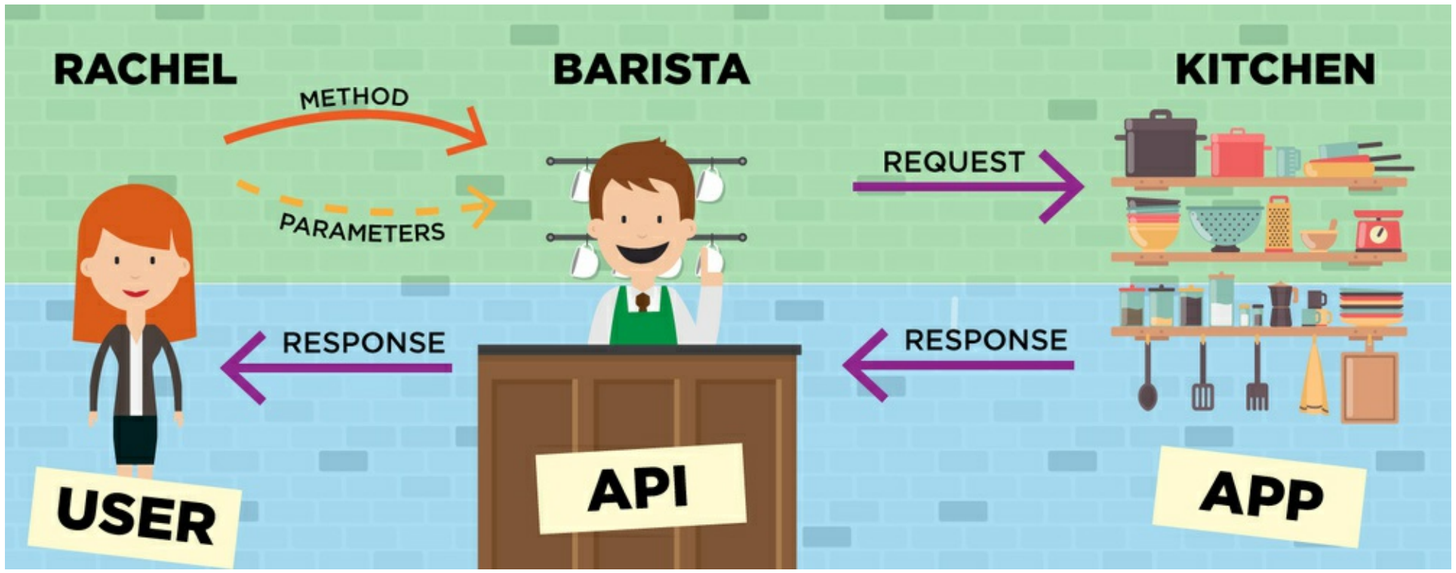
- Manual - easy
  - But not repeatable

- Automated - tricky
  - Hard to create
  - Hard to maintain
  - True end-to-end test
    - The Holy Grail!

  - When does a change mean a fail?
  - Fixing a usability problem
    - Intended change
    - Fails the test - fix the test

DevOps and Testing - API Testing

What's an Application Programming Interface?



- Turn a resource into a service
  - Contract - Set of calls, responses
  - Backend changes? OK - keep the API
  - System independence - isolation
- Protect the resource
  - Authentication/Authorization
  - Volume-limiter
  - Logging/Auditing

APIs in Modern Apps

- The interface between the front end and back end
  - Isolate backend from the front-end
  - One API multiple front-ends
    - Web
    - iPhone
    - Android

- An interface for other applications
  - Define the service - API
  - Expose the service for others
    - Bus Schedules/Locations
    - Court Lists
    - Access to legacy system e.g. ICBC
    - Manage Outlook appointments

Example API Call

```
GET api/moti/regions

[
  {
    "id": 200000,
    "ministryRegionID": 1,
    "name": "South Coast"
  },
  {
    "id": 200001,
    "ministryRegionID": 2,
    "name": "Southern Interior"
  },
  {
    "id": 200002,
    "ministryRegionID": 3,
    "name": "Northern"
  }
]
```

DevOps and Testing - API Testing

- Make an API call
- Verify ("Assert") the response is expected
  - Assertion not met - test fails

- Easy to write - text
  - Can even be generated - BDD

- Relatively easy to maintain
  - APIs are stable, so are the tests
  - Challenge - getting consistent data

- Visibility/Transperancy

DevOps and Testing - Non-Functional Requirements

- System response time is acceptable
- System is secure
- System can scale
- System is accessible (supports users with disabilities)
- System runs on all platforms (e.g. mobile, web browers, etc.)
- System has a disaster recovery plan
- System deployed properly
- System is online

Traditional - verify as an event during development

Pipeline - verify continually as the system evolves

Monitoring - verify continually as the system operates

Common Non-Functional Testing

- Load Testing
  - Typical: Use API (or UI) Testing from a scaled up external source
    - Script of "typical interaction(s)"
    - Tool/Service runs many instances of script - 100/500/1000, etc.
  - Usually an event to establish production baseline
    - Once in production - monitor production
    - However - you can still be hit with a "Day After" issue
  - In pipeline, run on non-prod environment to verify trend
    - Is the load test result the same from deploy to deploy?
  - Visibility/Transperancy

Common Non-Functional Testing

- Security testing
  - Static - analyze the code for quality and vulnerabilities
  - Static - verify libraries/versions against database of vulnerabilities
    - "You are using node.js version v5 which has vulnerability XYZ"
    - Like a virus checker, the vulnerability database evolves daily
    - Must run the scanning regularly
  - Dynamic - tool runs script of known hacking techniques
    - Call API with parmeters known to invoke vulnerabilities
  - Visibility/Transperancy
- Chaos Monkey testing

Monitoring your system in Production

Your Application doesn't end with the launch

Google's Golden Signals

- Latency - response time for requests
- Traffic - current activity on the system (requests/second)
- Errors - rate of requests that fail
- Saturation - current capacity available (CPU, memory, etc.)

Monitoring and Responding

- Monitoring usage trends:
  - Health Check - is the application running?
  - Average response time (request, API, database)
  - Request error rate
  - Resource usage vs. capacity

- Visibility/Transperancy

- Notifications of "out of range" events
  - Emails of all unexpected server errors

- The ops runbook
  - If this happens then do that
  - Leads to automating responses to events - no human intervention
    - e.g. Server crashed? Automatically restart new instance

The Application Health Check

Deployment Pipeline

- Is there an End-to-End **automated** deployment pipeline?
  - Does it include environments? Even devs?

- Are the manual steps and governance documented?
- How long does it take to deploy a trivial change?
- Is there a culture of continuous improvement?

Deployment Platform

- Is the platform flexible: resilient, scalable, extensible?
- Is the platform managed?
- How are platform events handled/escalated?
- Is there a culture of continuous improvement?
- Is there an Operations Runbook?

The Application Health Check

Application Independence

- Is the application isolated from other systems?
  - Who has to know if the system changes?
  - What has to happen if other systems are changes?

- Does the business own the deployment decision
- Is there a plan for eliminating dependencies?

The Application Health Check

Manual Testing

- Are the developers using TDD?
- Is there a tight develop/test relationship?
- Is there a thorough test plan?
  - Is there a framework for tracking test runs?

- Is pre-launch production load testing needed?
  - Large user base with the potential for usage spikes

Automated Testing

- How much testing is included in the deployment pipeline?
  - Unit Tests
  - Integration Tests
    - API Tests
    - User Interface Tests
    - Pre-promotion Load Testing
  - Security Scans
    - Static
    - Dynamic
  - Deployment Validation Tests
  - Visibility/Transperancy

The Application Health Check

Monitoring and Responding

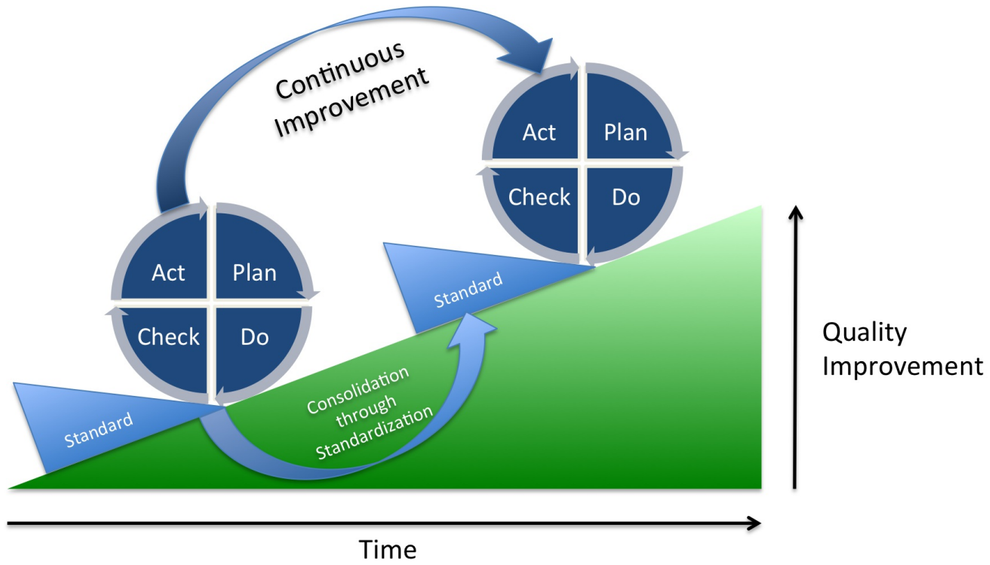
- Is there usage/performance monitoring?
  - Response time
  - Request rate
  - Error rate
  - Resource usage
  - Visibility/Transperancy
- Are error event notifications sent out?
- Are backup/restore processes in place?
- Are trends monitored?
- A **verified** disaster recovery plan?
  - e.g. What if the database is lost?
- Is there an application Ops Runbook?
  - What has to be done manually?
  - Are repeatable processes being automated?
  - Is there a culture of continuous improvement?

The Application Health Check

Focus on the pain

Provide feedback: Visibility/Transperancy

Establish a culture of continuous improvement



Review

Why is DevOps?

- We looked at:
  - Complexity in deploying applications using documentation and manual processes
  - Application Architecture
  - Why traditional approaches are hard for Devs and Ops
  - Why traditional approaches are prevalent in Government - project-focus

What is DevOps?

- We learned:
  - A culture of applying Lean principles to the end to end systems
  - Reduce risk by addressing it as early as possible - "shift left"
  - Backed by lots of powerful tools - largely developed and shared in the open
  - Lab - deploying a complex Web Application - easily

Review

What is the Cloud?

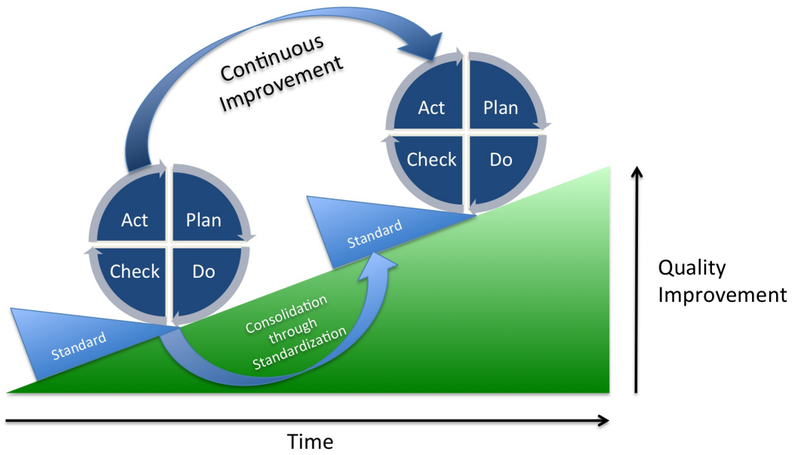
- We discovered:
  - About Data Centres
  - All the "as a Services" (and perhaps some digressions...)
  - Options for delivering applications on platforms
  - In BC - HPAS is the main choice, but more options are coming
  - Regardless, the underpinnings will be DevOps
    - Declare what you need - let the platform figure it out

Review

What happens on the ground?

- We learned about:
  - Extending the deployment pipeline
  - Testing types and best practices
  - Visibility/Transperancy
  - Application health check

Establish a feedback-based culture of continuous improvement



That's it!

Course Feedback: https://goo.gl/6qb4yl

References list available for those interested in learning more