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

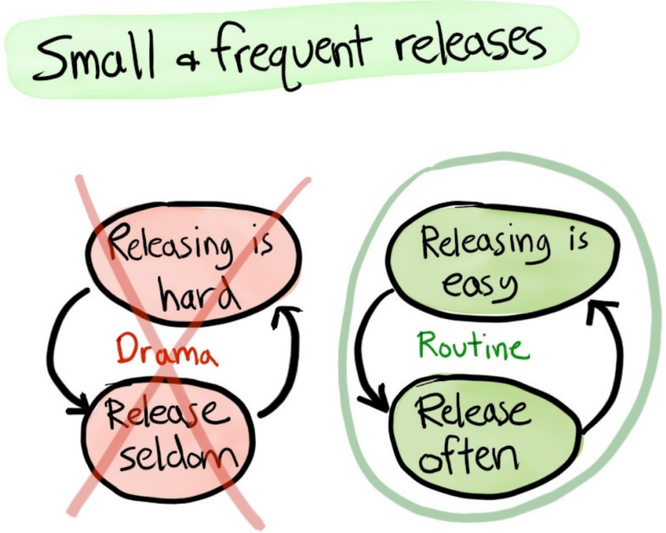
Part 2: In The Clouds, On The Ground

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

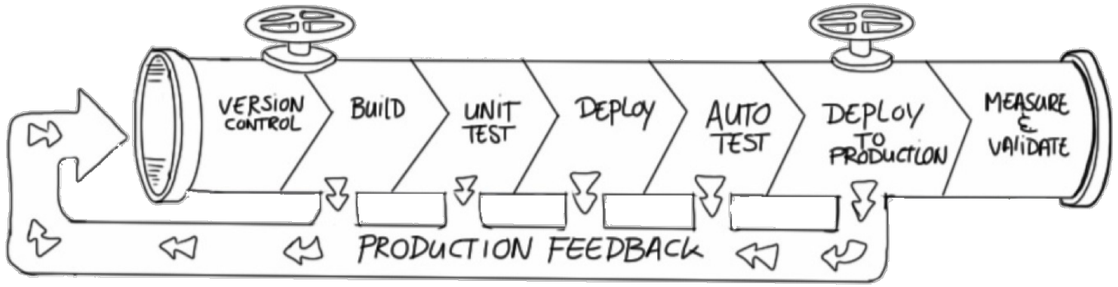


DevOps

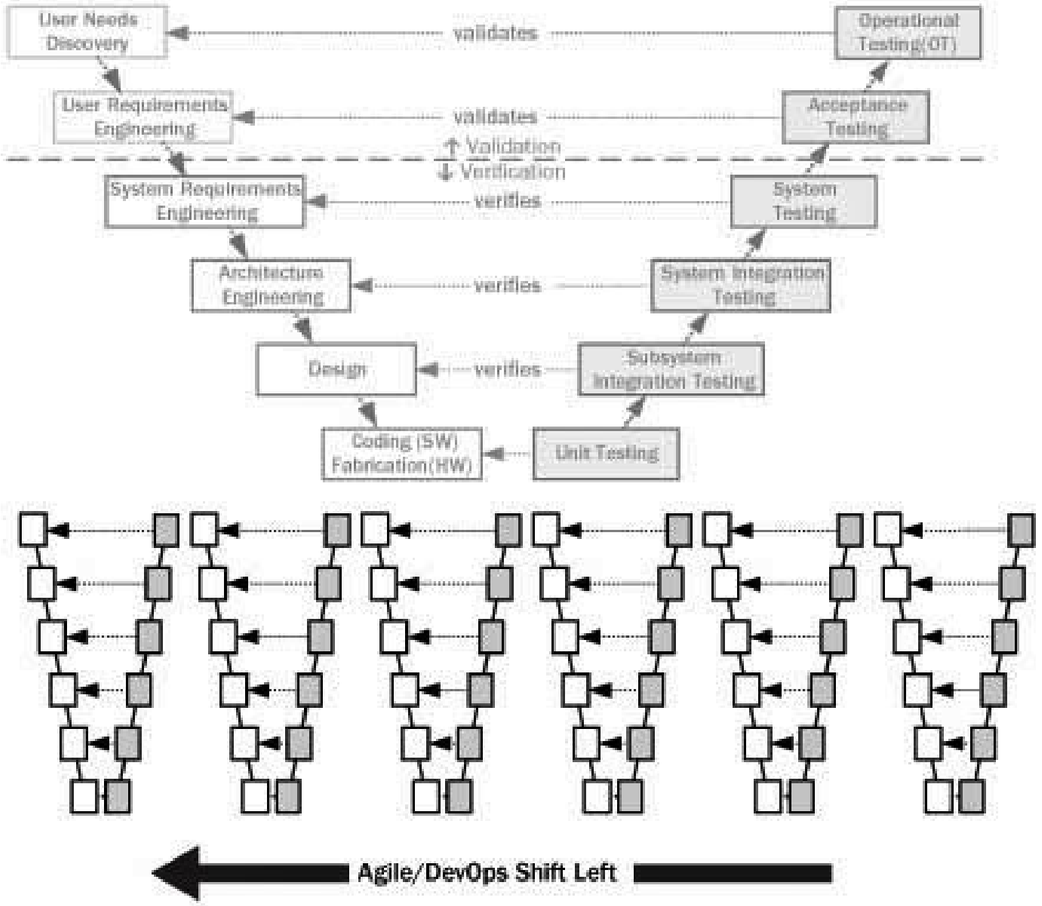
- The application of Lean Principles across the end-to-end system
  - A culture of continuous improvement
  - ...based on a the build up of powerful tools

- 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

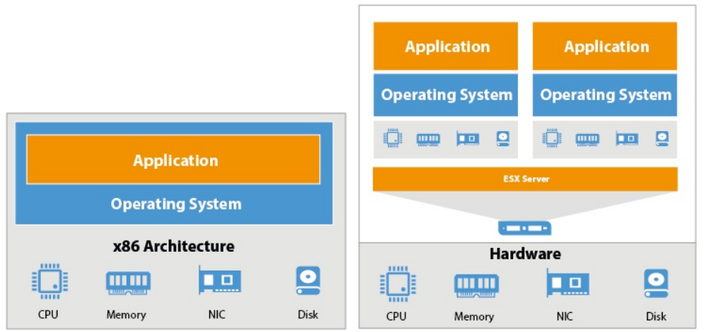
- Day After is just like the Day Before

What is the Cloud?

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

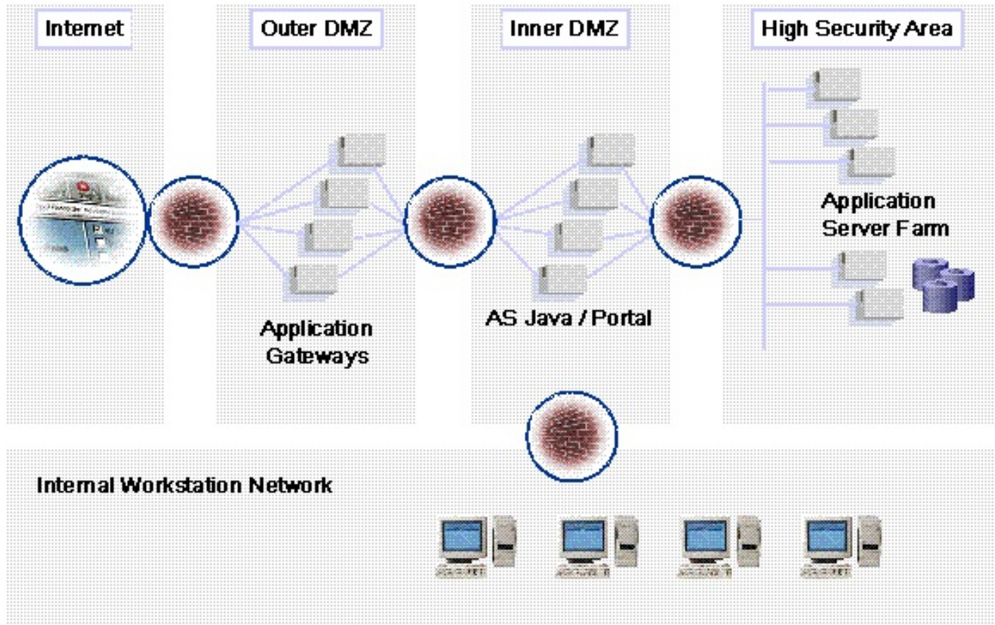
- Often discussed at the same time as DevOps - "Let's just run it in the Cloud"
  - What's the connection?

The Building Blocks of the Cloud



Physical vs. Virtual Machines (VMs)

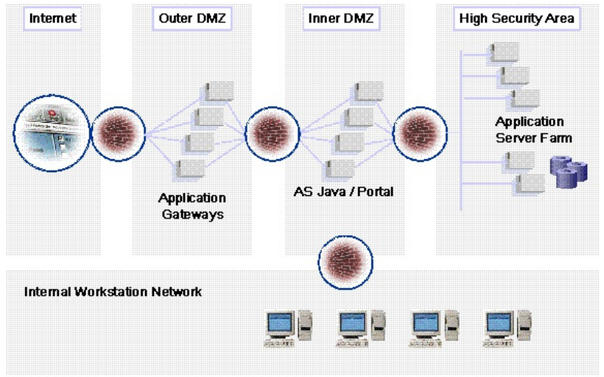
Networking and Security - pre-Cloud

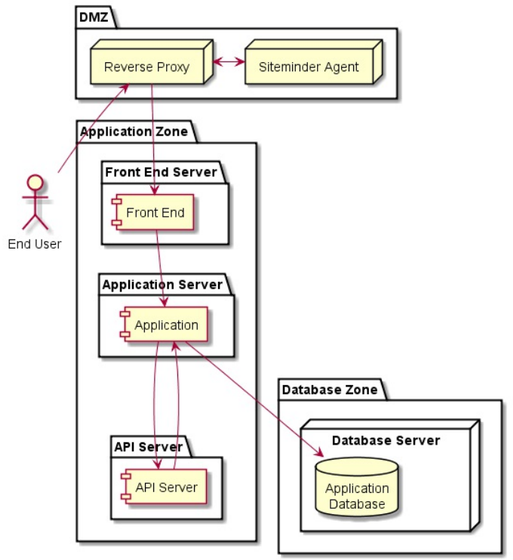


Times 3 - Dev, Test, Prod

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

The (Other) Cloud: * as a Service

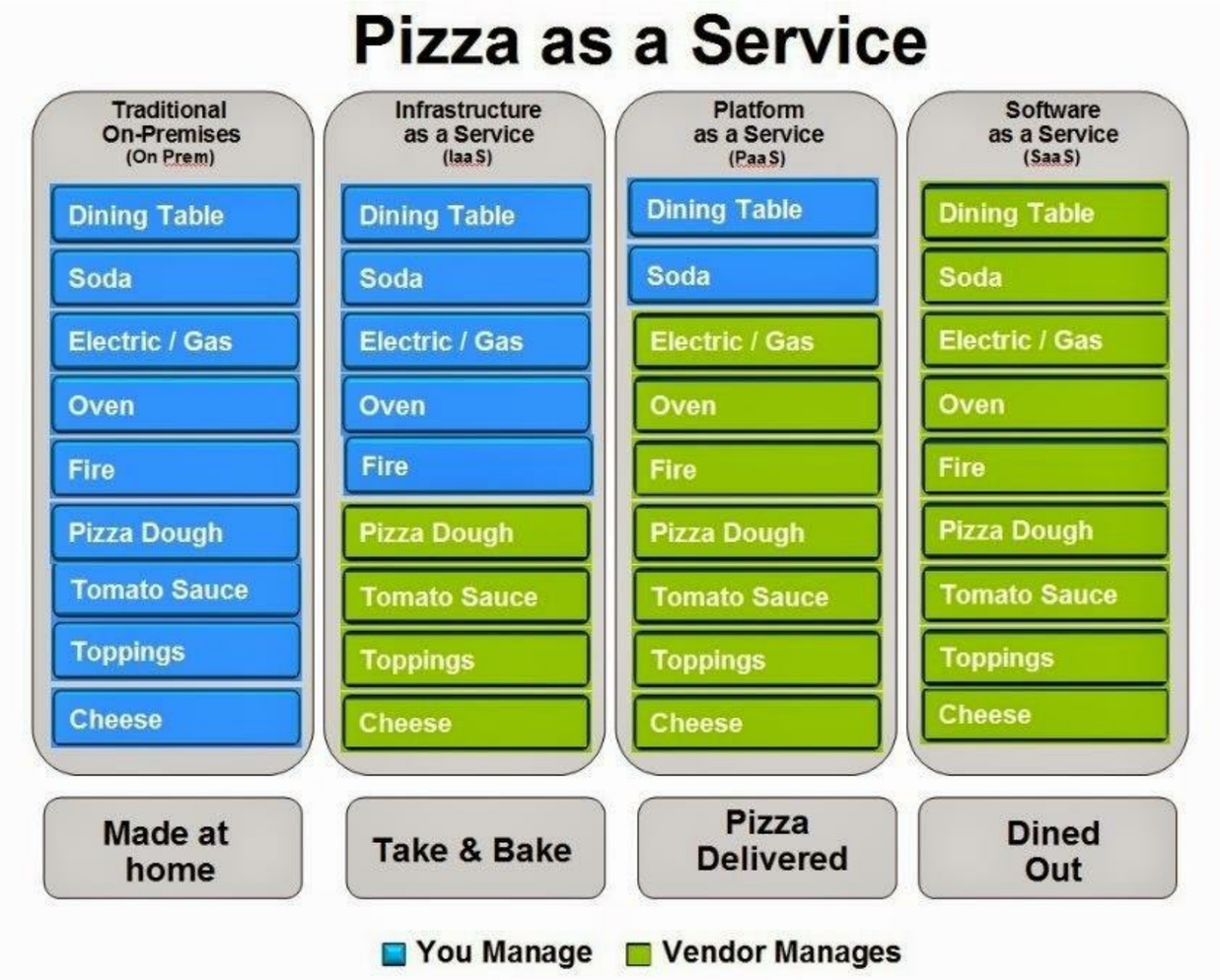
Once you aren't using your computer for computing purposes - you are using the Cloud

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

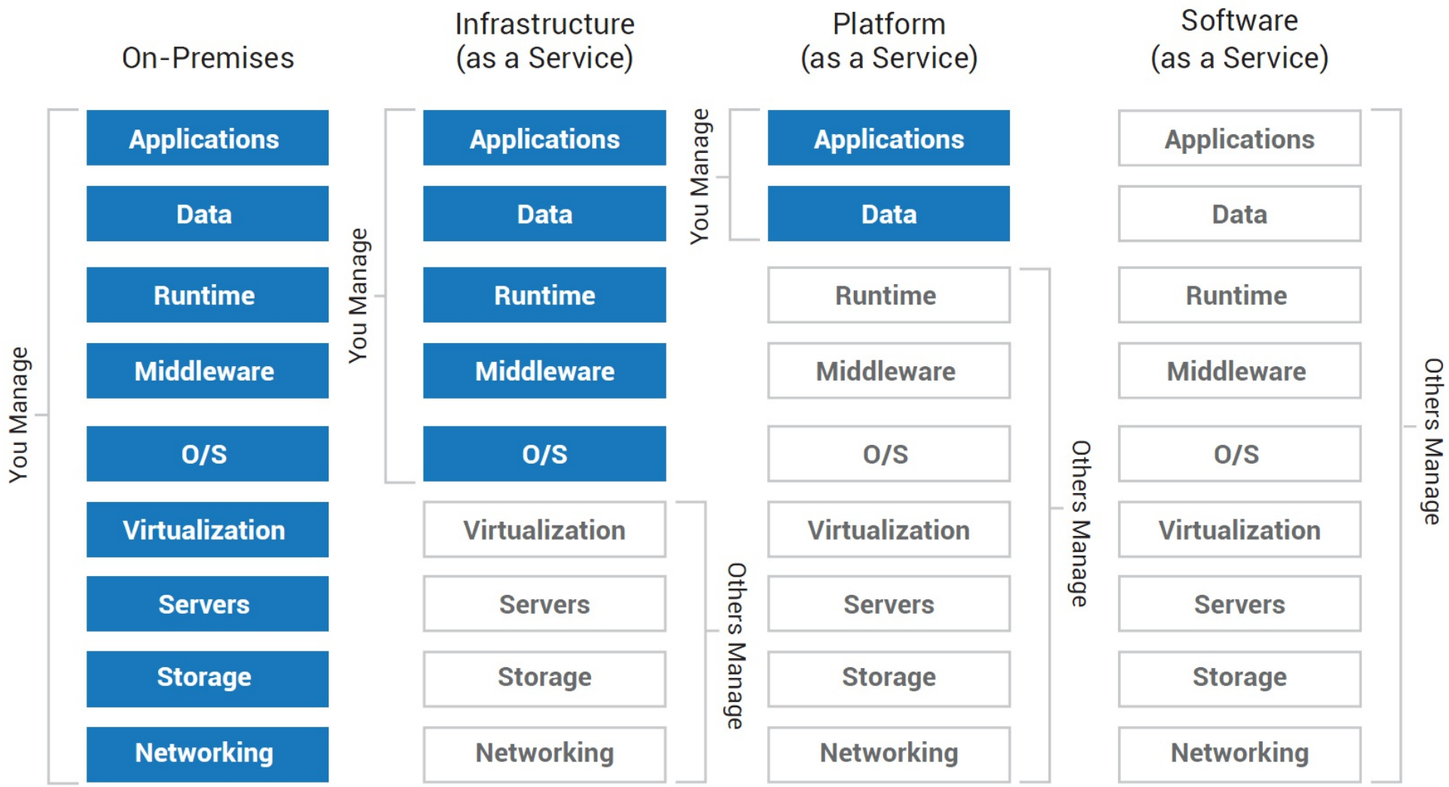
Since all gov't apps aren't on our computers - they are all "in the Cloud"

So let's talk * as a Service - IaaS, PaaS, SaaS

Pizza as a Service

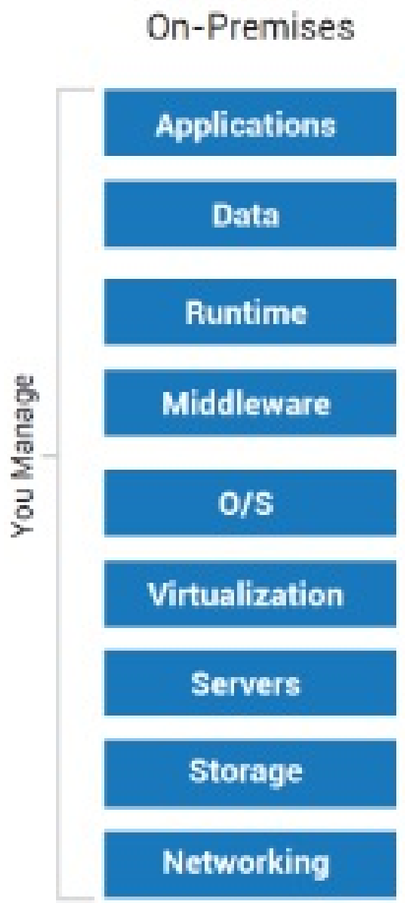


On-Premise, IaaS, PaaS, SaaS



HPAS Services / BC Gov't Data Centres

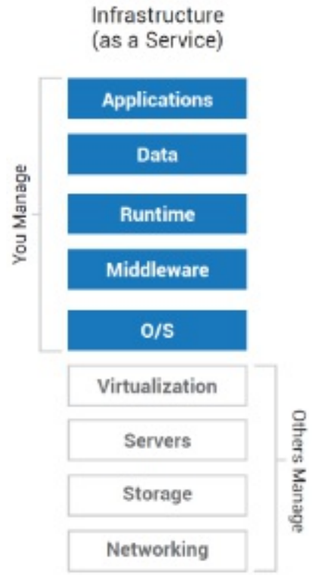
- On-premise-ish, with iStore orders
- Physical and VM servers
- Networking via VLANs and Firewall changes
- Standard images - Windows, Linux
- Standard software - e.g. Oracle, .NET
- No containers (Docker)
- Extra services - patching, monitoring
- Backups and restores
- New! IaaS-ish Offering
- New!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

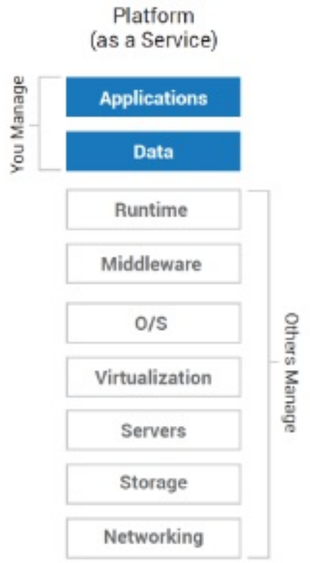


* And a lot more services...

PaaS: HPAS OpenShift, AWS, Azure

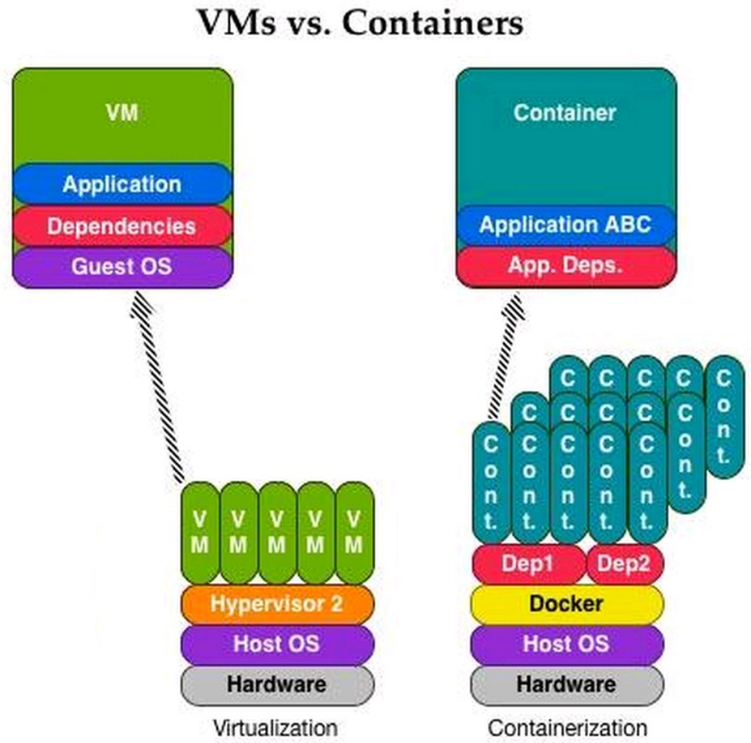
- Declarative needs - concise
- Dynamic setup and configuration
  - Spin up app components
  - Software Defined Network (SDN)
  - True cattle - die (and get replaced)
  - Cattle are less stable - that's Ok

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



Digression! Physicals, VMs, Containers

Goals: Resource optimization, consist execution - same everywhere



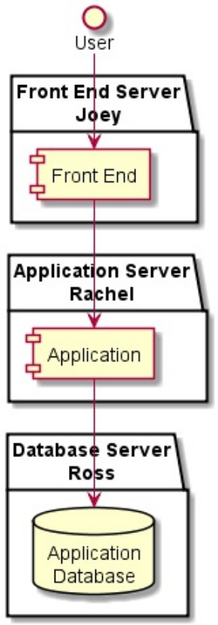
Container Benefits

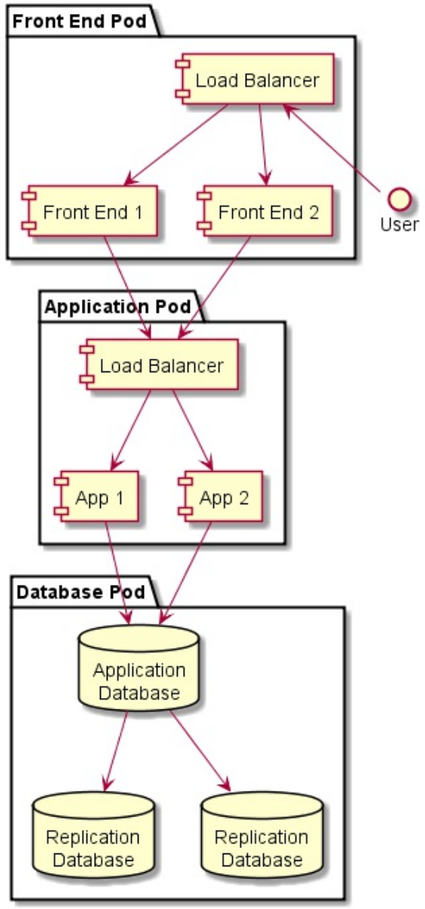
- Easy isolation - *feels like a whole computer*
- Really lightweight - high density - many on one host
- Create once, run everywhere - the "Shipping Container" analogy
  - Run the **same** container on dev machine and Production
  - Drastic reduction in shared dependencies

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?

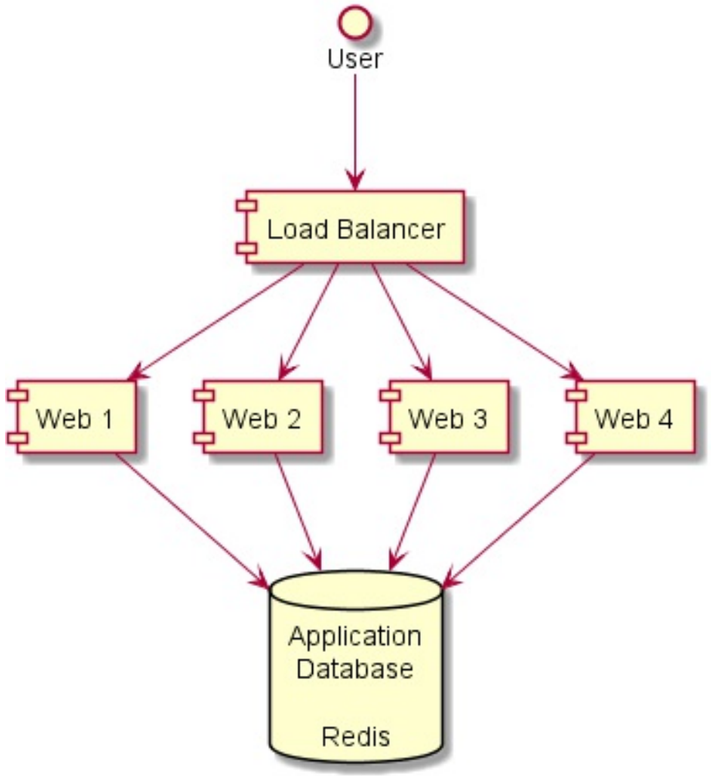




- What if *Front End 1* node crashes?
- What if the load goes up? down?
- What if the main database crashes?
- What if we want to deploy an update?

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
```

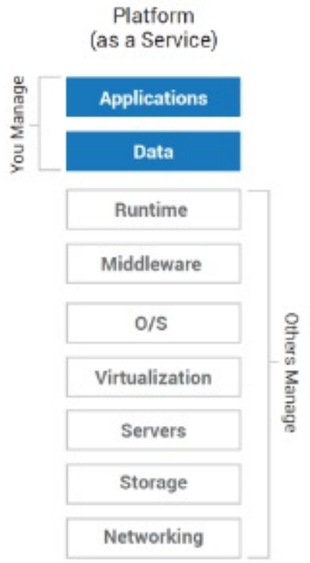


- With all networking connections defined

PaaS: HPAS OpenShift, AWS, Azure

- 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

- 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 make requirements too complex for other approach

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 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
  - Executed pre-commit to prevent regressions
  - Verification during/post-build
  - Product Owner: Visibility/Transperancy
    - Are we adding unit tests?
    - Are the tests passing?
- Integration Test - program units in combination
  - Example: app server code that updates the database
  - Created during the development process
  - Executed post-build
    - To verify the build
  - Can be defined using **BDD**
    - Behaviour-Driven Development
    - Suite of executable user stories
- 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
  - Automate - 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!

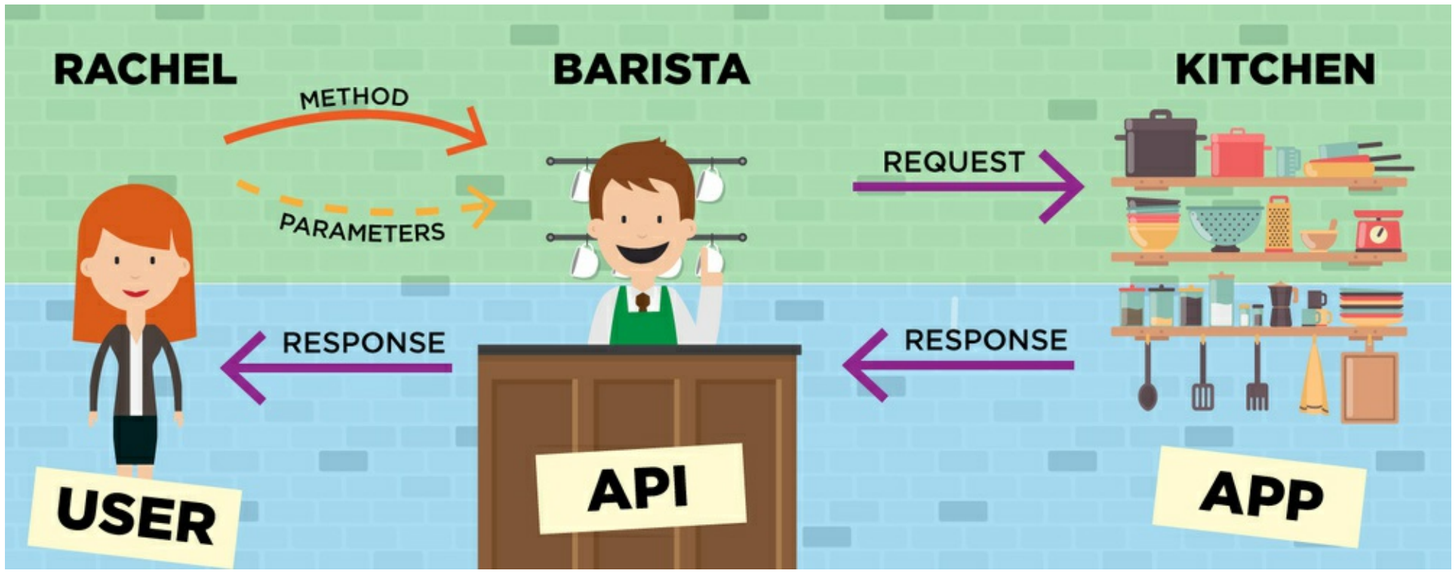
- What is easy?
- Where do they struggle?

- 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 - 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 - 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 - Non-Functional Requirements

- System response time is acceptable
  - Even with lots of users

- System is secure
- System can scale
- System is accessible (supports users with disabilities)
- 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

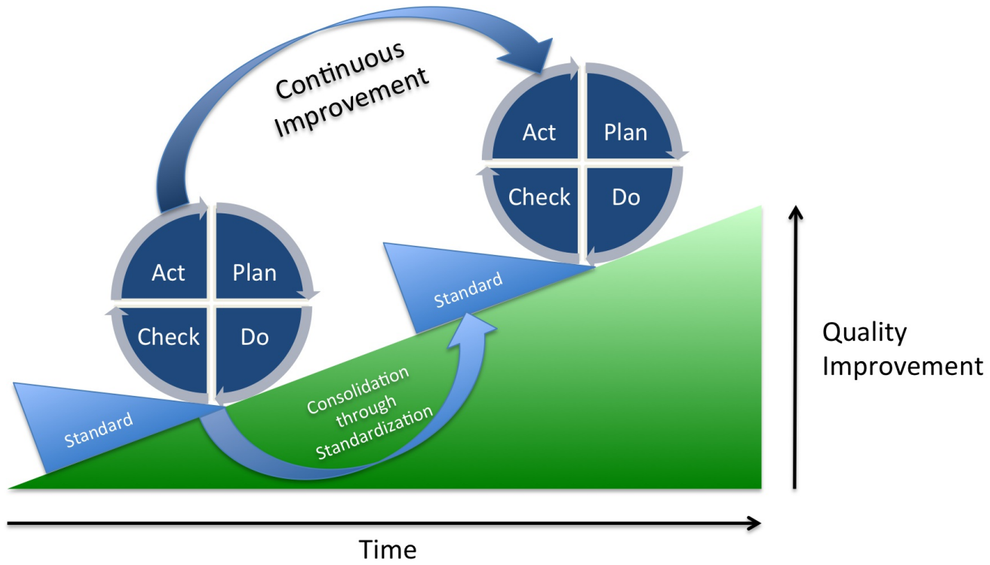
- Is there usage/performance monitoring?
  - Response time
  - Request rate
  - Error rate
  - Resource usage
  - Visibility/Transperancy
- Are error event notifications sent out?
- Are backup/restore processes in place?
- Are trends monitored?
- A **verified** disaster recovery plan?
  - e.g. What if 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?

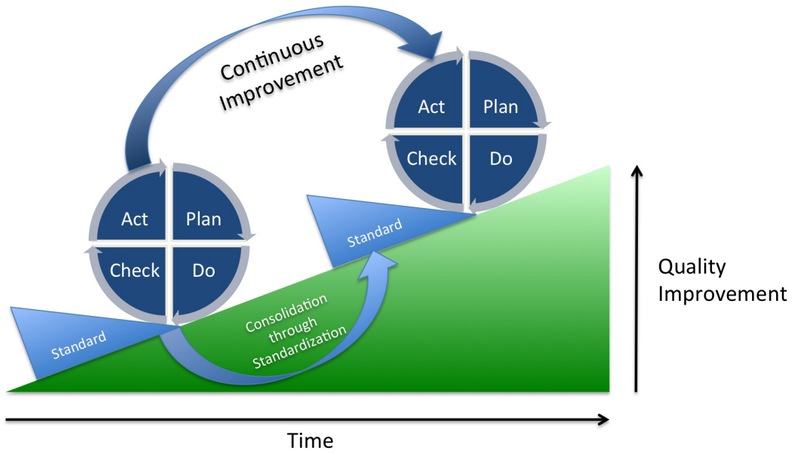
- We discovered:
  - About Data Centres
  - All the "as a Services" (and perhaps some digressions...)
  - Options for delivering applications on platforms
  - In BC - HPAS is the main choice, but more options are coming
  - Regardless, the underpinnings will be DevOps
    - Declare what you need - let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