# AP/ITEC 2210 3.0 A: System Administration Fall 2023

**Instructor: Jamon Camisso**
**ITEC 2210 Chat: mattermost.itec2210.ca**
**Email:** jamon@yorku.ca
**Website**: https://eclass.yorku.ca/

**Date/Time: Wednesday, 19:00-22:00**
**Location: Zoom / ACE 003**
**Office hours: Via Mattermost any time**

# Class 10 - Infrastructure as code

- Announcements
- Course Evaluations:
  - https://courseevaluations.yorku.ca/
- Final Exam:
  - 11 Dec, 19:00, room - DB 006

(ノ°□°)ノ ⌒ ┻━┻

# Class 10 - Infrastructure as code

- Infrastructure as code / configuration management

> “The code we use to control our infrastructure is not just part of our infrastructure, it is our infrastructure”
> PSNA p.55

# Class 10 - Infrastructure as code

- Infrastructure as code / configuration management
- My reductive version:
  - Don’t change a system, program systems
  - System Administration (these days) should be more about automation and programmatic interaction with infrastructure than hand crafting a web or database server configuration
  - Code and data determine how an environment is provisioned and operates

# Class 10 - Infrastructure as code

- Infrastructure as code / configuration management
- A few core principles:
  - **Machine-processed definition files**
  - **Document using code**
  - **Use version control (VCS)**
  - **Continuous Integration/Continuous Delivery (CI/CD)**
  - **Small batches principle - many small batches of changes**
  - **Continuous availability - strive for no downtime**

# Class 10 - Infrastructure as code

- Infrastructure as code / configuration management
- A few core principles:
  - Machine-processed **Definition Files**
    - Changes are made to config management files, not to servers
    - Configuration files define what a system should look like, execution engine makes it so
    - If you have to login to a host to make a change you're fired!

# Class 10 - Infrastructure as code

- Infrastructure as code / configuration management
- A few core principles:
  - **Document using code**
    - Use IaC manifests to document what a system should look like
    - How environments are built
    - Diagrams, documentation grow stale quickly

# Class 10 - Infrastructure as code

- Infrastructure as code / configuration management
- A few core principles:
  - **Use version control (VCS)** for IaC manifests
    - Source code, templates, static files, and data
    - Who made a change, when, why
    - Easy auditing and roll back if there are problems

# Class 10 - Infrastructure as code

- Infrastructure as code / configuration management
- A few core principles:
  - **Continuous Integration/Continuous Delivery (CI/CD)**
    - Test each change on commit/push - take it live if it passes (CD)
    - Helps with catching errors early on
    - Allows simulating changes to live environments

# Class 10 - Infrastructure as code

- Infrastructure as code / configuration management
- A few core principles:
  - **Small batches principle** - many small changes cf. a few large sets
    - Small changes make small bugs
    - Many small changes means less blocking others work
    - Reverting is easier when a small change can be cherry-picked

# Class 10 - Infrastructure as code

- Infrastructure as code / configuration management
- A few core principles:
  - **Continuous availability - strive for no downtime**
    - Allows more frequent changes
    - Enables small batches principle
    - Also ensures good architecture and environment design

# Class 10 - Infrastructure as code

- Infrastructure as code / configuration management
- Pets and cattle analogy from PSNA CH 3
  - Servers can be pets, each one a unique and special snowflake
  - Or like cattle, with thousands that are similar enough to be treated the same way, or in groups according to functionality
    - Group based on common characteristics - white, brown, dairy
    - Servers can similarly be grouped - web, database, firewalls

# Class 10 - Infrastructure as code

- Infrastructure as code / configuration management
- Neither is necessarily better or worse, but one approach is a lot more work when it is time to scale up, or make changes, or teach new hires, or …
- You will probably start out making pets as you learn about different tools and systems. That's OK! Trick is to do it again, and automate it
- However many times, a pet project that you intend to automate just works, goes live, and the automation never gets finished. Try to automate, don't let anything go live unless it can be programmatically duplicated

# Class 10 - Infrastructure as code

- Infrastructure as code / configuration management
- **Benefits** of IaC are in three main areas:
  - **Cost** - manual labour is reduced, or only incurred for up front design
  - **Speed** - tasks can be completed quickly, and in parallel
  - **Risk** - changes can be tested in dev or beta environments first

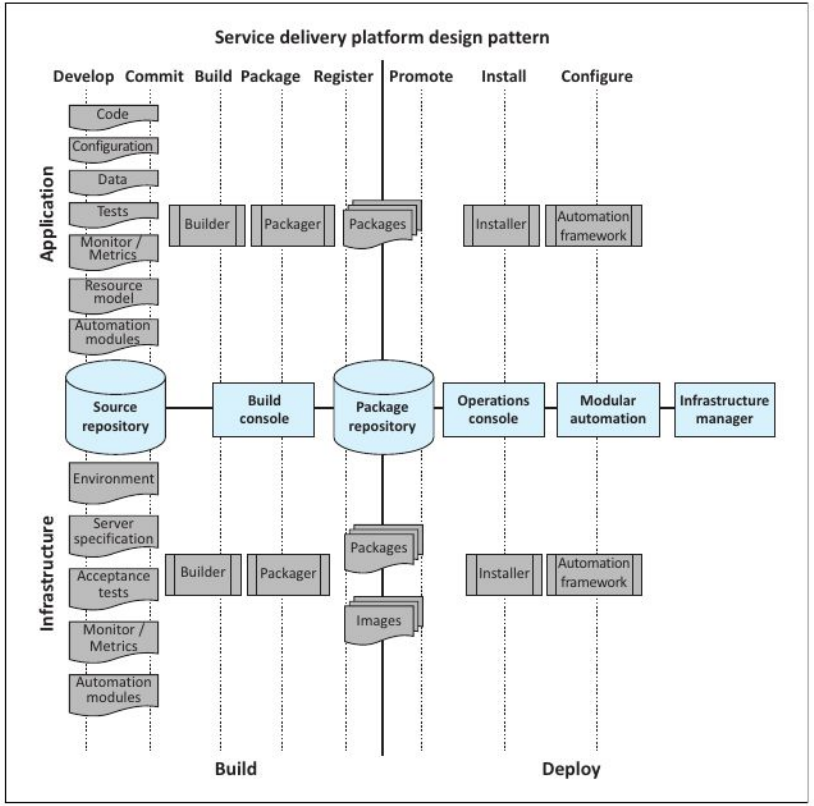
# Class 10 - Infrastructure as code

- Infrastructure as code / configuration management
- **Other benefits**:
  - Environments are **visible** to others, not just SAs
    - Better cross-team collaboration
    - Better on-boarding new hires
    - Makes people accountable for decisions
  - **Security & risk audits** can focus on config management, not checking every single system

# Class 10 - Infrastructure as code

- Infrastructure as code / configuration management
- Bug detection benefit - IaC allows for:
  - **Smoke tests** - does it work? Basic syntax check/linting
  - **Unit tests** - run portions of code, verify outputs match inputs
  - **Integration tests** - try running code in a pared down environment
  - **Acceptance tests** - customer can verify functionality with live data
  - **Load tests** - run benchmarks, look for performance regressions

# Class 10 - Infrastructure as code



# Class 10 - Infrastructure as code

- Infrastructure as code / configuration management
- Idea is to get infrastructure (bottom half) to look like software (top half)
- Packages and images can be cloned, version controlled, distributed and installed just like software
- Tools like Ansible, Puppet, CFengine, Salt, Docker, Kubernetes, Mesos make this easy
- Combined with a CI system like Jenkins, Gitlab CI, Travis CI etc. infrastructure and applications can be deployed & tested together

# Class 10 - Infrastructure as code

- Version control systems

Listing 4.1: Git annotate output

```
ChangeId:  (Author:   Date:      Line:)   Code:
eae564f2   (craigp    2015-08-06   172)   isRelayed := r.Header.Get(relayHeader)
67b67bd0   (mjibson   2015-04-20   173)   reader := &passthru{ReadCloser: r.Body}
67b67bd0   (mjibson   2015-04-20   174)   r.Body = reader
67b67bd0   (mjibson   2015-04-20   175)   w := &relayWriter{
67b67bd0   (mjibson   2015-04-20   176)           ResponseWriter: responseWriter}
ba83ae2e   (craigp    2015-09-16   177)   rp.TSDBProxy.ServeHTTP(w, r)
ade02709   (ipeters   2016-02-23   178)   if w.code/100 != 2 {
ade02709   (ipeters   2016-02-23   179)           verbose("got status %d", w.code)
67b67bd0   (mjibson   2015-04-20   180)           return
67b67bd0   (mjibson   2015-04-20   181)   }
67b67bd0   (mjibson   2015-04-20   182)   verbose("relayed to tsdb")
a6d3769b   (craigp    2015-11-13   183)   collect.Add("puts.relayed",
a6d3769b   (craigp    2015-11-13   184)           opentsdb.TagSet{}, 1)
```

# Class 10 - Infrastructure as code

- Version control systems (VCS)
- Idea is each line and change can be attributed to someone, but can also be reverted or modified as needed
- Example looks like a Golang code to integrate with **OpenTSDB**
  - If I had to guess, I'd say it is for monitoring & alerting integration
- Point is, commit logs and annotation show who did what, when & why

# Class 10 - Infrastructure as code

- Version control systems (VCS)
- Even in places where there are explicit peer reviews to changes, changes will get missed
- Such changes are affectionately known as ‘cowboy’ changes. Imagine:
  - “I cowboyed a change to a sysctl setting to allow tracking more connections”
  - 2 years later: “Why doesn’t this new server that’s 5x better hardware, running a clone of the old server’s disks fall over with only half as many connections as the old system?”

# Class 10 - Infrastructure as code

- Version control systems (VCS)
- Cowboys happen in the worst places - storage, DNS, firewalls, routers
- See also PSNA B.2.4 (short read)
- To avoid issues, any cowboy change should be captured in a VCS, even if it is just local to the system in question
- As simple as ‘git init /etc/’

# Class 10 - Infrastructure as code

- Configuration Management
- Various open source systems already mentioned:
  - Ansible, Puppet, Chef, CFengine, Saltstack
- Use **Declarative** manifests or languages to achieve a desired final result
  - CM system can be imperative or declarative behind the scenes, you just care about declaring what a system should look like
- Many use **Domain Specific Language** (DSL) to achieve final result

# Class 10 - Infrastructure as code

- Configuration Management
- Idea with **Declarative** systems and **DSL** is you define what something should look like, **not** the steps to make it happen
- CM system will compile or interpret your manifests and run OS and application specific modules or commands to achieve final goal
  - All will know yum on Redhat/Fedora/Centos, apt on Debian/Ubuntu
  - Most will have a module for MySQL, PostgreSQL, sqlite, etc

# Class 10 - Infrastructure as code

- Configuration Management
- **Idempotent:** CM system will only make changes if needed
  - Code can be run a million times, actual change or operation will only run if it is needed
    - Can be thought of as if/else conditional
    - If Apache is not installed, install it
    - Else (implicit Apache is installed), don't install it (do nothing)

# Class 10 - Infrastructure as code

- Configuration Management
- **Idempotent:** CM system will only make changes if needed
  - Harder to implement than it sounds but very important to get right
    - Ensures all changes are predictable
    - Efficient - most CM runs will result in no changes, fast to run
    - Allows chaining actions based on when changes happen

# Class 10 - Infrastructure as code

- Configuration Management
- **Idempotent:** CM system will only make changes if needed
  - Also makes it much harder to leave a cowboy change in place!
  - For example, if puppet is set to run every 15 minutes and someone made a change, it would be reverted back after 15 minutes at most
    - Useful to ensure cowboys get reverted, but also (sometimes) to make and test changes and get back to a known state

# Class 10 - Infrastructure as code

- Configuration Management
- **Guards** and **Statements**
  - Guards check to see if a change exists on a system
  - Statements are the code that make the change (e.g. your manifest)
  - Guard is run before and after a change to ensure a statement has been run successfully (in the past, or during current CM invocation)
  - Guard re-invocation is key to idempotency - checks same condition

# Class 10 - Infrastructure as code

- Configuration Management
- **Dry-run** modes
  - With guards, a change can be evaluated in terms of what it will look like, without running the actual statement
  - Just run a guard to check for a desired state - if it doesn't exist, the set of needed changes is known
    - (puppet freebie: puppet agent --test --noop (optional --debug))
    - (ansible freebie: ansible-playbook --list-tasks)

# Class 10 - Infrastructure as code

- Configuration Management
- Probably best to not roll your own. If you do how will you:
  - Handle concurrency?
  - Deal with errors or failures?
  - Make changes to the CM architecture itself?
  - Ensure secure communication channels?
  - Monitor the CM system itself?
  - Test the CM system?
  - Teach others how to use it?
  - Add new features?
  - Most big CM systems out there have dealt with these issues***

# Class 10 - Infrastructure as code

- Deploying IaC/CM in an organization
- Start small, (even at home with a laptop or desktop computer)
- Get in the habit of defining what your system should look like - treat any change you're making directly on a system as a cowboy change
  - Manage individual files or directories to start
  - Add in desired packages, services, data
  - Scale to managing small parts of many systems
  - Version control your manifests/changes
  - Eventually use an OS image with just enough to load a CM agent

# Class 10 - Infrastructure as code

- Deploying IaC/CM in an organization
- Once you have a base image with an agent baked in, integrate with CI/CD
- Have your CI system (Jenkins, Travis CI, Gitlab CI etc.) provision a system and then bootstrap the CM agent
- Once you have an instance running with an agent in CI, run tests against CM manifests to test changes
- From there, automate environment creation, add containers, orchestration

# Class 10 - Infrastructure as code

- Collaboration benefits
- Peer review - hopefully benefit is self-evident here
  - (also lets you share responsibility for bad changes!)
- Helps mentor new system administrators or other team members
- Enables self-service options, where SAs just approve changes from developers instead of having to make changes from start to finish

# Class 10 - Infrastructure as code

- Downsides & Myths
- Steep learning curve for some systems
  - Thousands of excellent online tutorials
  - Many books on each system
  - Start small, you'll become an automation fanatic very quickly once you get a feel for how easy changes can be versus manual configs
  - Making ethernet cables really isn't fun, automation is

# Class 10 - Infrastructure as code

- Downsides & Myths

- Dangerous, risk of a sorcerer's apprentice situation
  - With VCS systems, even far reaching changes can be rolled back
  - Automated tests (CI) can prevent a runaway situation from occurring

- Manual configuration is better
  - Automated linting, peer review, testing make this a non-issue

- Automation will put you out of a job
  - This should be your goal! There's always more work to be done, try to make yourself totally redundant with automation