* What do you want to accomplish with DevOps?
  + increase value given to customers
    - customers enjoy things that we give them, so they stay customers
  + respect for your people
    - make you people feel valued effectively
* LEAN
  + focus on customer value
  + eliminate waste
  + one-piece flow
  + shared continuous learning
  + eliminate overburden of people
  + theory of constraints
* C.A.L.M.S.
  + **Culture**. People and process at the forefront
  + **Automation**. Repeatability by automating activates
  + **Lean**. Appling lean principles to I.T.
  + **Metrics**. Measure extensively and use to improve
  + **Sharing**. Continuous learning by sharing knowledge

Making a DevOps transition

* Changing the Culture
  + Re-establish your purpose
    - what are your *shared* objectives?
    - does everyone empathize with the customer?
    - are you going through the agile motions?
    - misalignment here will doom your effort
  + Elevate empowerment
    - trust the team, let them take ownership
    - employees do what is needed run the service
  + Promote accountability
    - can't have empowerment without accountability
    - quality built into the process
    - shared commitment to excellence
    - reward who deliver and take responsibility
  + require teamwork
    - all disciplines working together, in crisis or daily operations
    - respect unique skill sets
    - find excuses to get the team together
    - don’t reward the "lone genius" (that is *waste*)
  + encourage continuous learning
    - offer resources and time to learn
    - encourage experimentation
      * learn what makes the customer happy
    - conduct blameless postmortems
      * embrace failures as learning opportunities
    - make it easy to share and discover info
  + share, live, and reward the right values
    - don’t just say your values, LIVE THEM
    - lead by example
* Changing the organization
  + understand your customers and value streams
    - build up and value empathy for users
    - step back and "see" the system in place
    - perform value stream mapping exercises to focus your improvement efforts
  + put the spotlight on bottlenecks (constraints)
    - its ineffective to optimize anywhere but the constraint
    - a constraint can be internal or external
  + do away with inconsistent environments
    - applies to different environments
    - result of this bottleneck? slower software releases and issue resolution
    - aim for on-demand with no configuration drift
  + automate the path to production
    - how fast can you deploy code? is inventory piled up?
    - coding or testing faster won't help if deployment isn’t optimized
    - solution involves continuous integration and comprehensive, automated pipelines
  + build in quality up-front
    - QA teams aren’t responsible for quality
    - shared commitment to excellence means testing early and often
    - achieved through practices like TDD (test driven development) and continuous integration
  + break down communication barriers
    - poor communication results in push, not pull-based systems
    - feedback loops re critical to context and shared ownership
    - solved via a mix of tools, practices, a sincere intent to collaborate
  + create permanent teams around core products
    - teams don't disband after shipping v1, they build *and run* their service
    - the benefit? retain knowledge, continuously update services, improve customer satisfaction
    - not a fit for every product, but puts focus on customer value
  + to achieve velocity, you need complete teams
    - must be capable of going from lead to production on their own
    - encourage generalist skill sets versus hyper-specialization
    - may use services or APIs from constrained teams
  + eliminate waste in the organization
    - policies and procedures get heavier over time, return to the intent
    - automate manual steps wherever possible
* overcoming objections
  + objection: "DevOps makes us less secure"
    - focusing on quality that which reduces vulnerability
    - automation means fewer manual mistakes, and faster vulnerability resolution
  + objection: "we have offshore teams"
    - co-location of teams isn’t always possible or necessary
    - nature of outsourced partnerships has to change
  + objection: "you're eliminating our Ops team"
    - operations are *more* important, but role changes
    - it's about running platforms, creating APIs, delivering pipelines
  + objection: "I've got too much COTS and legacy code"
    - TDD, continuous integration, deployment automation possible with legacy environments
    - remember the point isn’t just pushing code faster, but improving the flow
  + objection: "we don’t have these skills"
    - an investment worth making
    - many of these are "soft" skills
    - make it group learning and apply immediately

Introducing DevOps Automation

* things to remember about DevOps and tools
  + the goal is continuous improvement
  + you cannot "buy" DevOps
  + tool adoption happens in stages
* the DevOps tool chain
  + planning, issue tracking, source control, build, testing, continuous integration and deployment, configuration management, cloud platforms, monitoring and logging, communication, knowledge sharing
* planning tools
  + shared purpose, transparency, empowerment
    - GitLab, tasktop, trello, azure boards, pivotal tracker
* issue tracking tools
  + customer responsiveness, limited knowledge waste, feedback loop
    - Jira, Zendesk, JetBrains YouTrack
* source control management tools
  + asset control, limit transportation waste, empower teams
    - git, GitHub, GitLab, Bitbucket, Subversion
* build tools
  + consistent packaging, automates an error-prone activity, yields early quality signals
    - Maven/Gradle, MSBuild, Rake, JFrog Artifactory, Sonatype, NuGet
* testing tools
  + puts focus on built-in quality, creates confidence on deployed artifacts
    - Junit, XUnit.net, Selenium, Jasmine, Cucumber
* continuous integration tools
  + fast feedback, reduce defect waste and waiting waste
    - Jenkins, CircleCI, TravisCI, Concourse, AWS CodePipelines
* continuous deployment tools
  + limit your pre-production inventory, automate complex pipelines, unify team around getting value to customers
    - Spinnaker, Octopus Deploy, AWS CodeDeploy
* configuration management tools
  + enforce consistency, treat infrastructure as code
    - Terraform, BOSH, Chef, Ansible, Puppet, Google Cloud Deployment Manager
    - treat infrastructure as a configurable automation-centric resource versus something you manually handcraft, deploy and manage yourself
* cloud platform tools
  + automation-friendly, observable runtimes for software
  + decentralizing, making it easier for other teams to use infrastructure and not have a blocker, bottleneck, or constraint
    - AWS, Microsoft Azure, Google Cloud Platform, Pivotal Cloud Foundry, Heroku, Container Schedulers
* monitoring and logging tools
  + seeing health of the entire system
  + fast recovery, responsiveness, transparency, limited human involvement during incidents
    - ELK Stack, Datadog, New Relic, Prometheus, Zipkin, Azure Monitor
* communication tools
  + connect teams, limit waiting waste, improve collaboration
    - Slack, Microsoft Teams, Google Teams, Zoom
* knowledge sharing tools
  + reduce knowledge waste, increase new-hire productivity, limit repeat mistakes
    - GitHub pages, Confluence, Jekyll, Google Sites, Wiki