SYNERGETICS
GET IT RIGHT

DevOps

Practice, process and toolchain

Microsoft Partner
Silver Learning
Gold Cloud Platform

Microsoft
Cloud Solution Provider

Educate Advise Implement Manage

Agenda

- What is devops?
- Why DevOps?
- DevOps Benefits
- Why should I care?

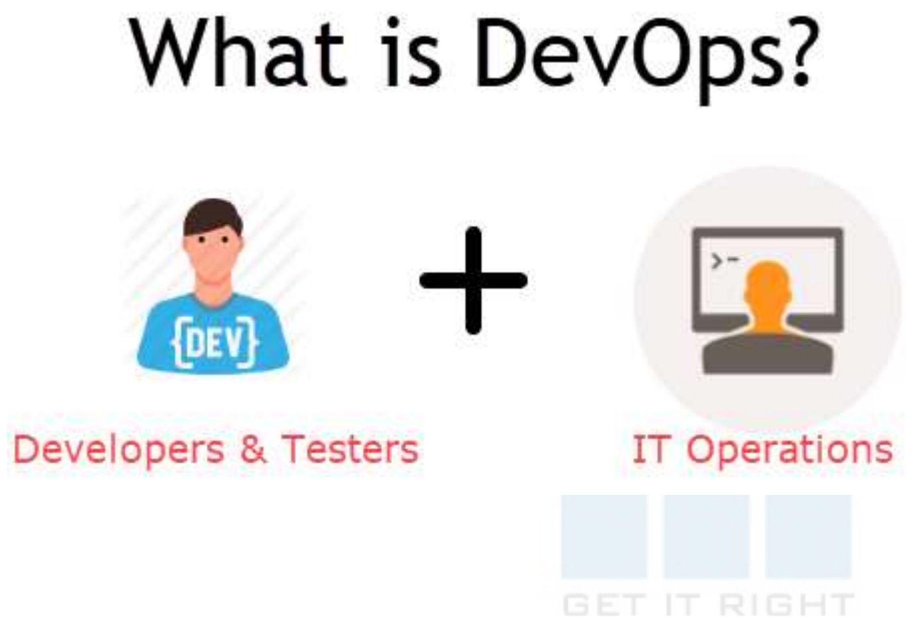
What is DevOps

DevOps is the application of agile methodology to system administration

Tom Limonceli (The practice of cloud system administration)

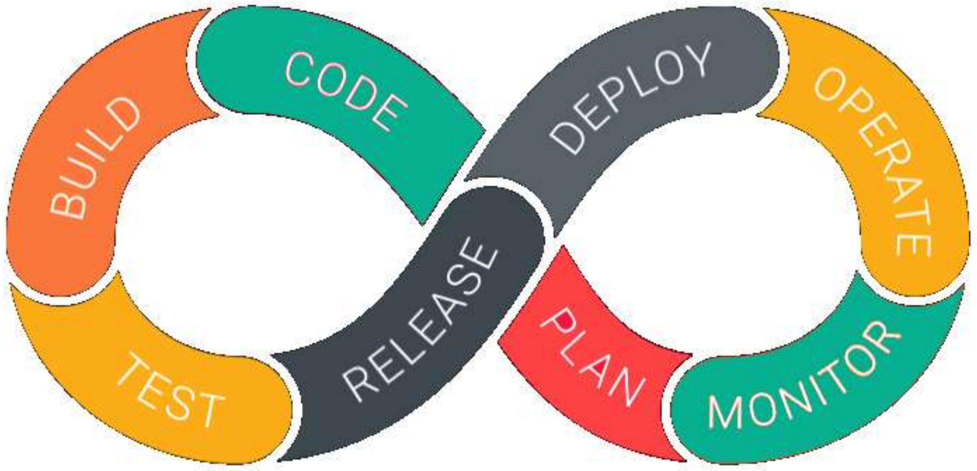
What is DevOps

- DevOps is the practice of operations and development engineers participating together in entire service lifecycle, from design through development process to the production support.



What is DevOps

- Culture : People > Process > Tools
- Automation: Infrastructure as Code
- Measurement : Measure Everything
- Sharing : Collaboration/Feedback



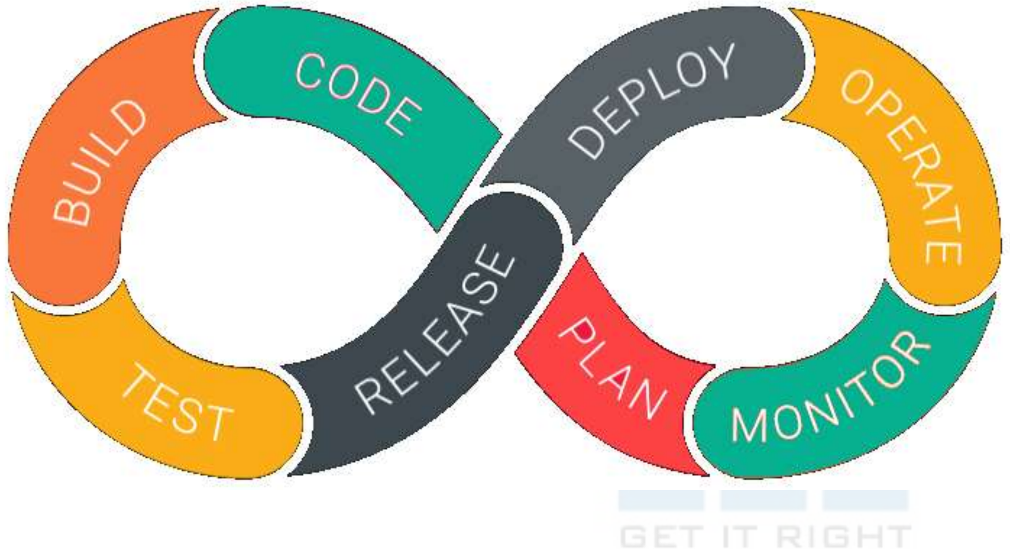
Why DevOps : The problems we face

- Defects are released into production, causing outages
- Inability to diagnose production issues quickly
- Problems appear in some environments only.
- Blame shifting/fingerpointing
- Long delays while dev, QA or another team waits on resource or response from other teams.
- Manual error is commonly cited root cause.
- Releases slip/fail.

SYNERGETICS
GET IT RIGHT

Why DevOps

- Demand for Faster delivery
- Outages are expensive
- Retaining talent



We want …. More **resilient systems** that easily **absorb change** from a **customer centric** organization that **attracts talent**.

Why DevOps

Its about using co-ordination and automation to deliver higher quality software, faster.

DevOps benefits

- Deploy 46x more frequently
- Offer 330x faster lead time for changes
- Deliver 96x faster mean time to recovery
- Posses a 5x lower change failure rate

State of Devops Report (2017)

DevOps core values

- Empowered individuals
- Accountability
- Teamwork
- Trust
- Transparency
- Continuous Learning
- Feedback loops
- Data driven decisions
- Standardization
- Customer centric

SYNERGETICS
GET IT RIGHT

Empowered Individuals

- Empowered within the system
- Do what’s needed to maintain the service
- Engineers can “pull the cord”
- Responsible for proposing improvements

Accountability

- Flip-side of being empowered
- Everyone responsible for quality, everywhere
- Part of “you build it, you run it” mentality
- Accountable to teammates as much (more so?) as management

SYNERGETICS
GET IT RIGHT

Teamwork

- No lone geniuses
- Coordination across functions at *all* times
- Respect for distinct skill sets
- Pays off during periods of crisis

GET IT RIGHT

Educate Advise Implement Manage

Trust

- Trust in each other, trust in management
- Requires everyone to be working off the same values and objectives
- Establish trust within and across teams
- Ok to “trust but verify”

Transparency

- Visibility into objectives, priorities
- Clear knowledge of long-term plans
- Shared access to metrics, tickets source code

Continuous Learning and improvement

- Requires introspection, identifying root cause of problems
- Sprint and incident retrospectives are great sources
- Learning should clearly make its way back into the organization
- Growing individual and team skills requires management investment

Feedback loops

- How are you learning about apps in production?
- Consider structured way to solicit feedback from *all* stakeholders
- Necessary to have fast-feedback channels available

Data-driven decisions

- Upfront data capture and analysis capability drives good decisions
- Not a replacement for thinking
- Drives consensus, prioritization
- Only the “right” metrics

Standardization

- Care about quality
- Prioritize repeatable actions through automation
- Scale through consistency
- Consensus use of standards

Customer centric

- Empathy for the customer
- Focus on value to the customer
- Cross-org, customer-driven goals
- Single definition of “done”

Pattern and practices

Value Stream mapping

- Getting a big picture view of the deployment pipeline
- Identifying value and waste
- Elevating bottlenecks
- Creating “flow”

Balanced teams, engaged management

- Balanced teams with no handoffs
- Favor generalist skills set, but don’t eliminate specialties
- Compress management hierarchy, break up functional silos
- Establish new workspace layout

Team Stand-ups

- Balanced teams with no handoffs
- Favour generalist skills set, but don’t eliminate specialties
- Compress management hierarchy, break up functional silos
- Establish new workspace layout

On-call engineering

- Rotate team members through on-call
- Responsible for service support
- Puts the focus on the customer
- Encourages instrumentation and continuous improvement
- Keeps the rest of the team focused

Agile planning of small batches

- Ship often, get fast feedback
- Product owner maintains backlog, team decides when sprint is “full”
- Tasks scoped to max of two days work
- Scope doesn’t change
- Team alwa

Continuous integration

- Daily check-ins to master branch
- Some form of TDD is critical
- Offers fast feedback, smaller problem surface to investigate
- “Green” builds are good, “red” should not be feared

Treat infrastructure as code

- All interactions with infrastructure done via source-controlled artifacts
- Make infrastructure work visible work
- Establishes standardized processes that are repeatable
- Eliminate wasteful snowflake environments

You build it, you run it

- Long-lived service teams vs. project teams
- Develop customer empathy, foster continuous improvement
- Helps prevent overloaded releases
- Improves stability of key services

Comprehensive telemetry

- No guesswork during service interruptions
- Heavily instrumented environments, with event-driven notifications
- Need baselines to identify anomalies
- Over-alerting causes fatigue

Blameless post-mortems (retrospectives)

- Not about assigning blame
- Understand the situation, timeline ahead of the meeting
- Use “Five Whys” technique to identify root cause, not just source
- Assign owners to action items

Standard tools and processes

- Avoid unique toolchain across teams
- Empower teams to suggest improvements to standard tools
- Automation is often, but not always, the answer

Continuous delivery, deployment

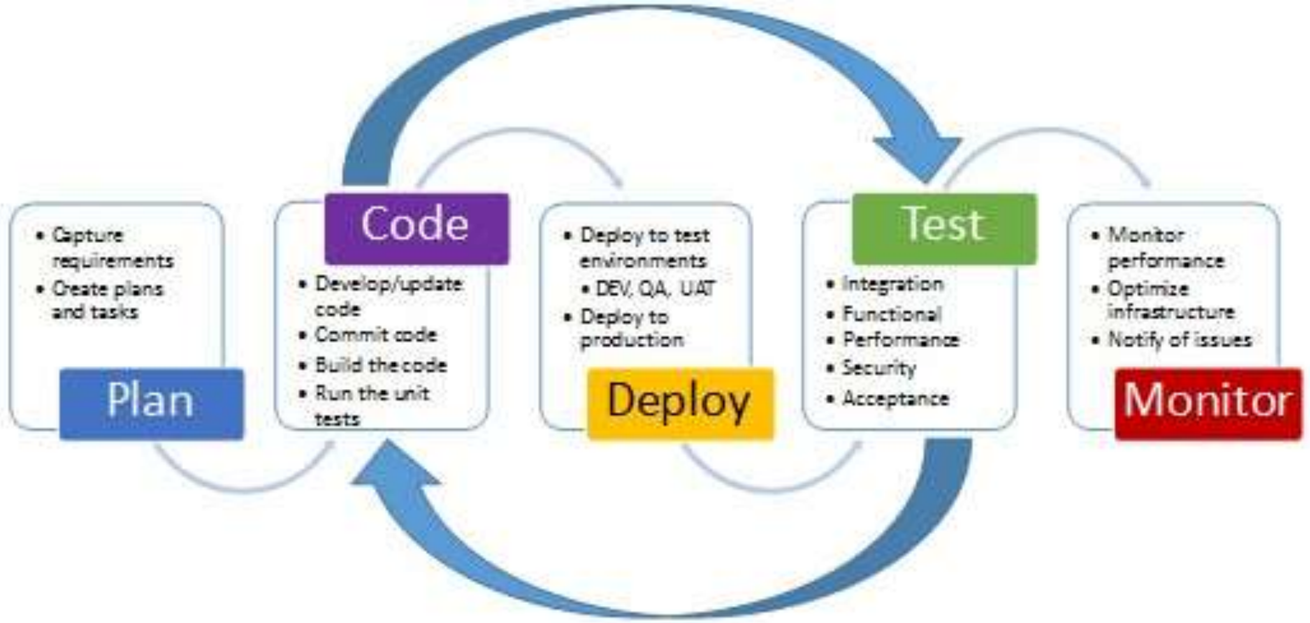
- Should be boring process that doesn't cause service downtime
- Consider what it takes to fully package software for deployment
- Partner with security and compliance groups
- Operations skills come in handy here

Show-and-tell session

- Help others learn from your success and failure
- Create possibilities for reuse across teams
- Visibility helps other teams coordinate their own milestones
- Encourages experimentation and continuous learning

DevOps Tool chain

- Collaboration
- Planning
- Source control
- Issue tracking
- Configuration management
- Development environments
- Packaging, artifact management
- CI / CD
- Application platforms
- Monitoring



Collaboration

- Connect your teams
- Improves situational awareness
- Chat rooms, knowledge repositories
- Quick collaboration prevents waiting waste

Examples: Slack, Microsoft Teams, GitHub Wikis

Planning

- Improved visibility of upcoming work
- Dynamic tools, not static documents
- Easy to see bottlenecks, breakdown in flow
- Also makes sense to include bugs, toil in planning view

Examples: Pivotal Tracker, Trello, Jira

Source Control

- Possibly the most important tool in a DevOps environment
- Store code, config, scripts
- Avoid transportation waste of moving code around unnecessarily
- Change history is important

Examples: Git, GitHub, GitLab, Bitbucket, Subversion

Issue Tracking

- Single way to collect, triage, and respond to issues
- Avoid info lost in transit
- Helps improve the feedback loop
- Product defects likely tracked elsewhere

Examples: ZenDesk, GitHub issues, Jira

Configuration Management

- Enforce state of compute resources
- Treat infrastructure as code, and avoid configuration drift
- Changes applied systematically
- Sometimes used as infrastructure build tools

Examples: Puppet, Chef, Ansible, Salt, CFEngine, BOSH

Dev Environments

- Consistent developer experiences
- No more “works on my machine”
- Mix of configuration management and supportive environments
- Cloud IDEs a fit here

Examples: Vagrant, Cloud9, Eclipse Che

Packaging, artefact management

- Typically the output of a CI process
- Improves flow, consistency
- Multiple ways to package an app
- Should package *all* corresponding components

Examples: Docker, JFrog Artifactory, BOSH

CI/CD

- Keep code “inventory” small and get fast feedback on code changes
- Call out to code scanning, quality components
- Visible status drives accountability
- Pipeline-based models are popular

Examples: Concourse, Jenkins, CircleCI, Bamboo

Application Platforms

- Modern runtimes that simplify deployment and ops at scale
- Provide APIs for deploying and managing apps
- Designed to reduce toil and automate lifecycle activities

Examples: Cloud Foundry, AWS Elastic Beanstalk. Azure App Service

Monitoring

- DevOps happiness may hinge on your approach here
- Need views of system health, not just server/app health
- Improves transparency, helps reduce MTTR

Examples: New Relic, Dynatrace, Datadog, ELK