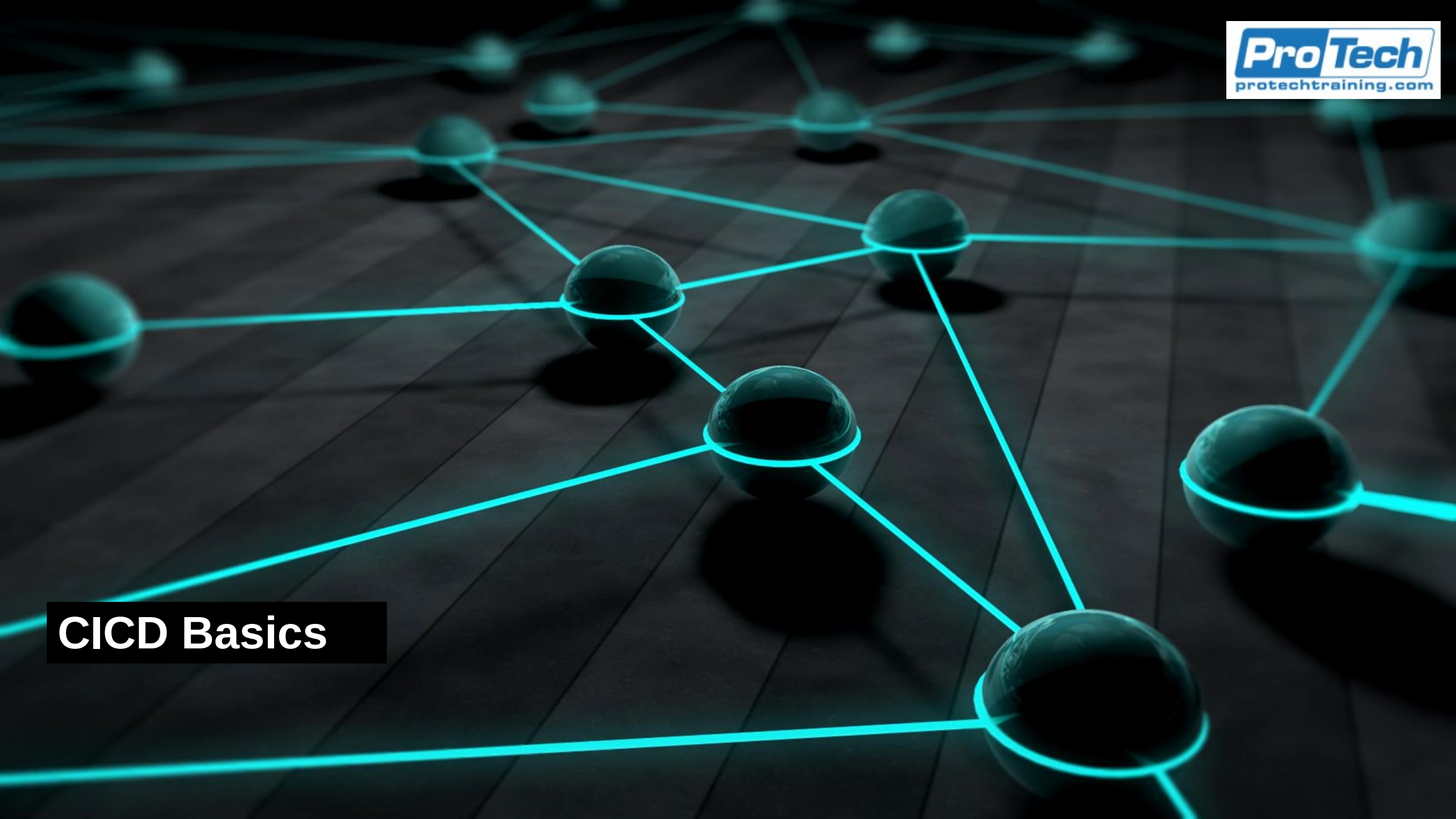


CICD Basics

Defining CICD

- CICD is not a methodology
  - It is not Agile or DevOps
  - Although both rely on CICD and use it extensively
- CICD is process automation applied to SE
- Similar to other kinds of automation (including robotic process automation)
  - The goal is to improve process efficiency and effectiveness
  - CICD is process agnostic
  - It can be used anywhere a SE process is well defined
  - Using CICD with bad processes makes them worse

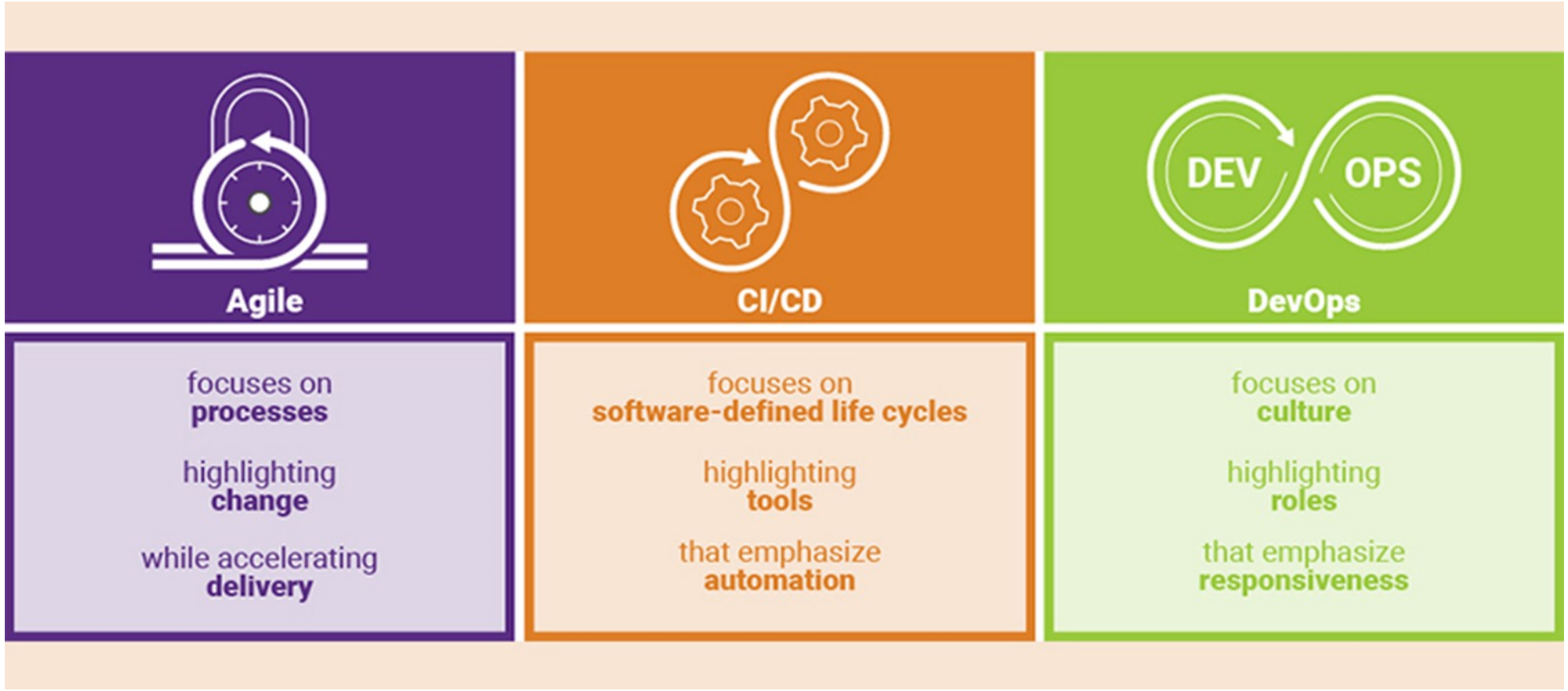
A fool with a tool is still a fool

Martin Fowler

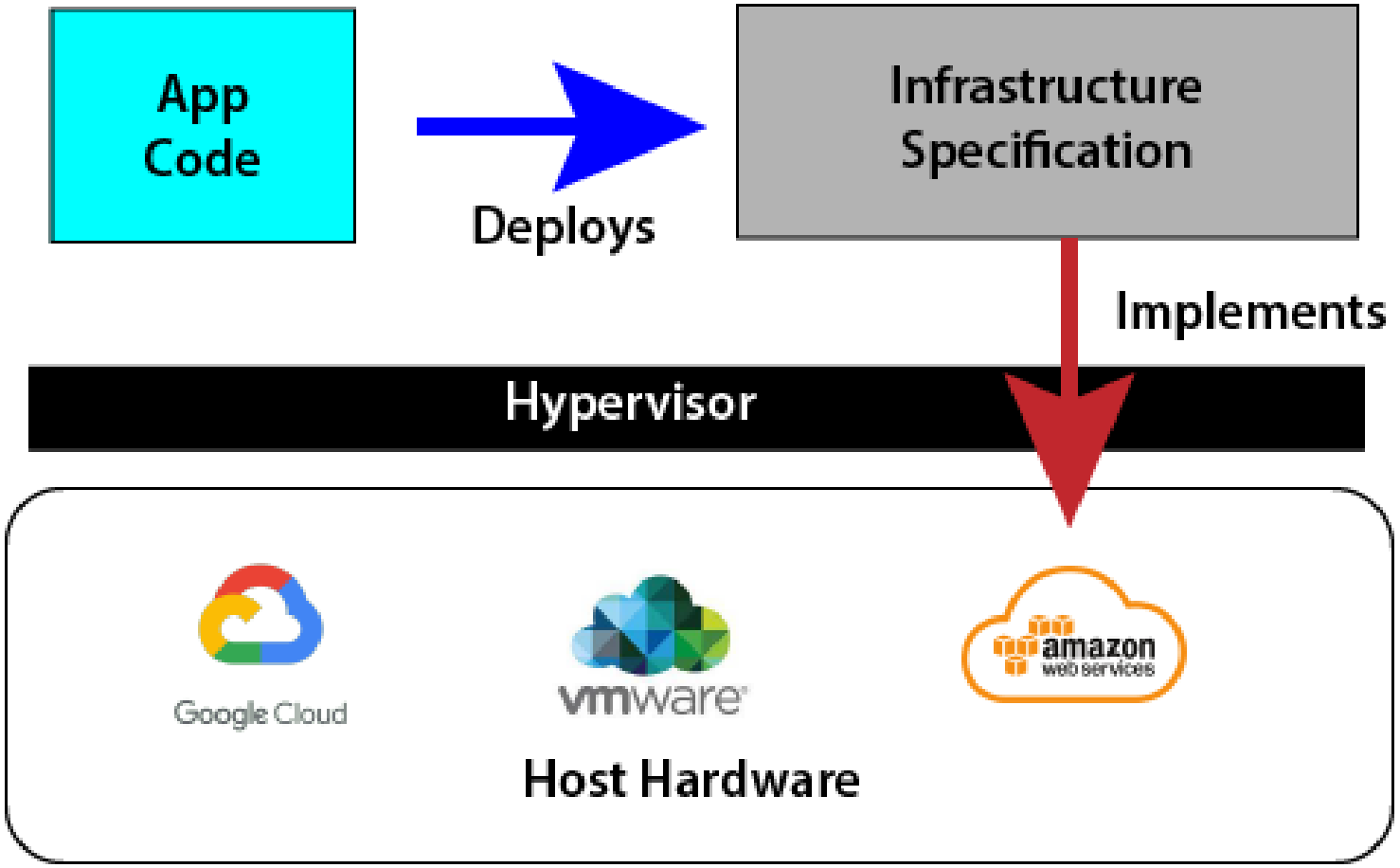
A computer lets you make more mistakes faster than any invention in human history – with the possible exceptions of handguns and tequila

Mitch Ratcliffe

Agile, DevOps and CICD



Infrastructure as Code

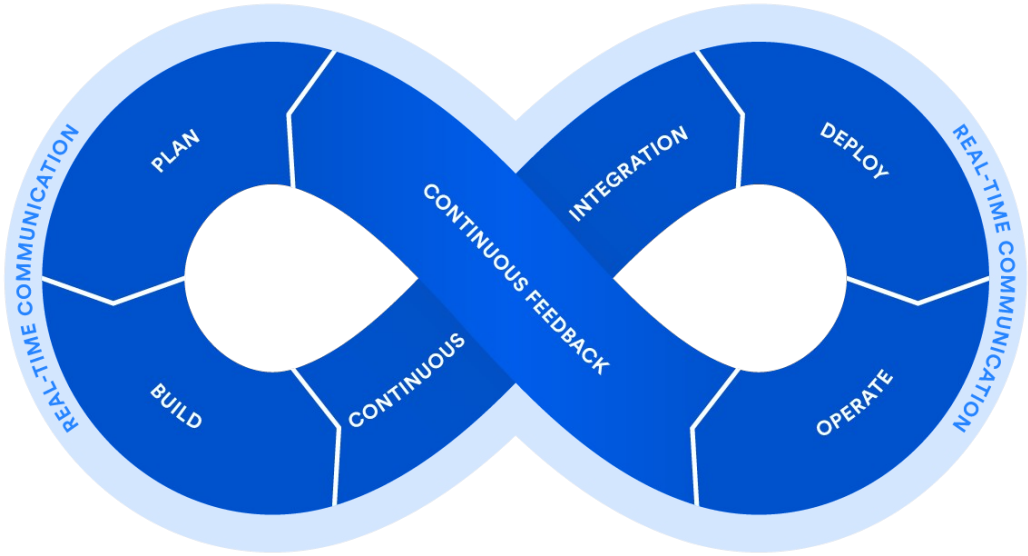


Agile, DevOps and CICD

- Agile
  - Focuses on iterative development, delivering small increments of value frequently.
  - Encourages collaboration between developers, testers, and business stakeholders.
  - Supports adaptability to changing requirements.
- DevOps
  - A cultural and technical movement bridging Development and Operations.
  - Emphasizes collaboration, automation, monitoring, and rapid feedback loops.
  - CI/CD is the engine that powers DevOps practices.
- CI/CD
  - Continuous Integration (CI): Developers frequently integrate code into a shared repository.
    - Automated builds and tests validate each integration.
  - Continuous Delivery (CD): Code is automatically prepared for release after passing CI.
  - Continuous Deployment: Extends delivery by automatically deploying changes into production after all checks.

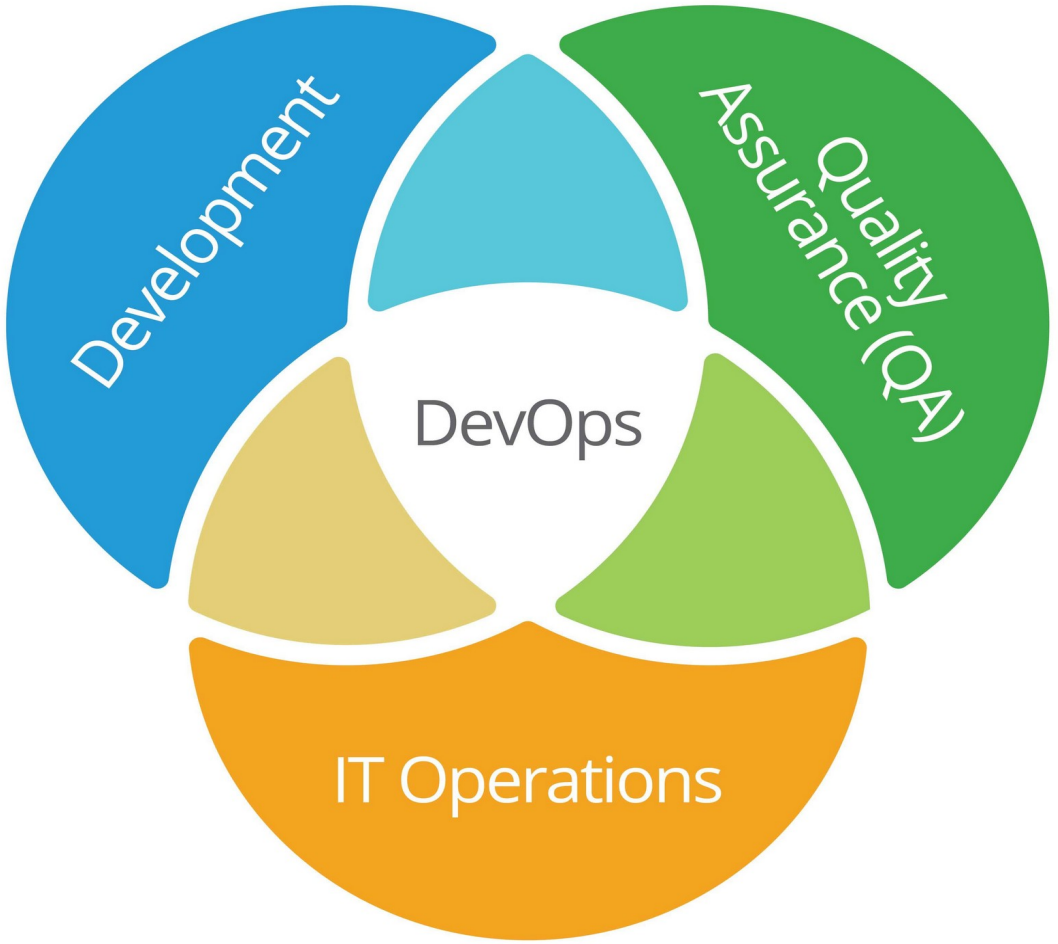
DevOps

- Driven by virtualization and Infrastructure as code
  - Dev and Ops had been two separate worlds
  - Dev was sort of automated
- Ops was manual and bare metal
  - Virtualization turned it all into code
  - Now the same tools can be used in the entire life cycle of a software product
  - Opportunity for full process automation support



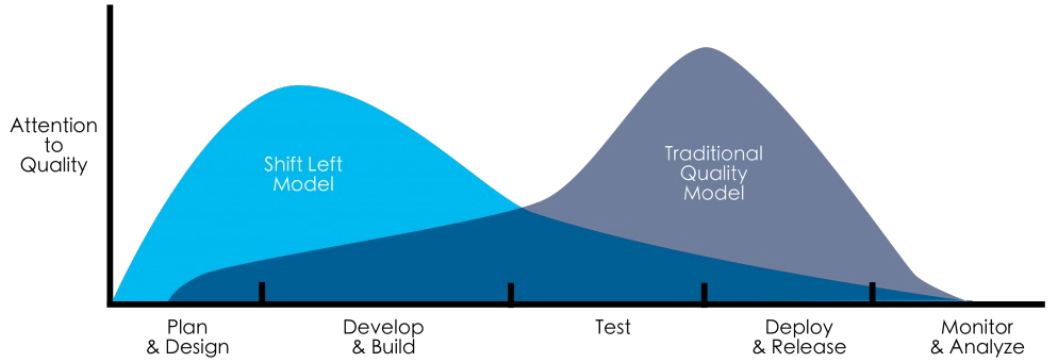
The Goal of DevOps

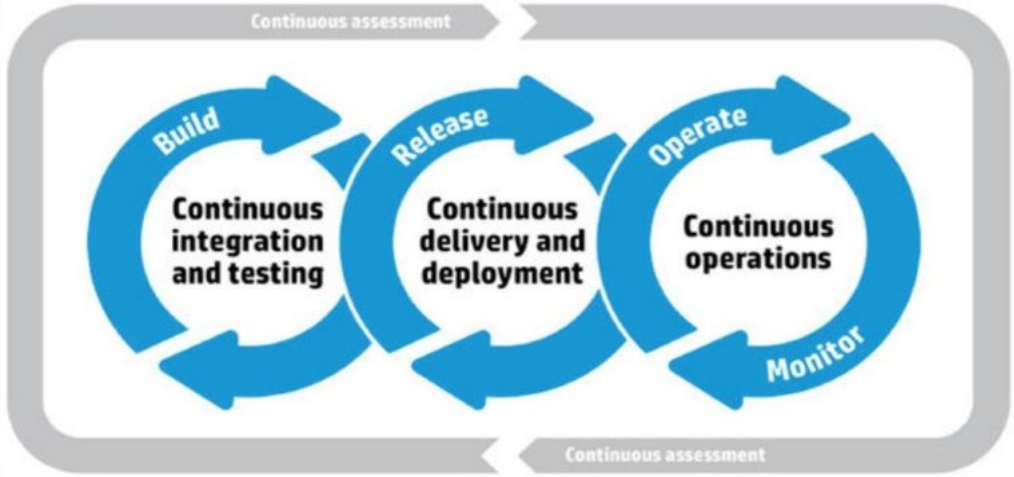
- Desilo-ize the three areas in software development
  - Get everyone using the same sorts of tools, practices and automation
- CICD
  - Continuous Integration: continuous integration of multiple development activities
  - Continuous Delivery: build artifact made available for delivery
  - Continuous Deployment: Delivered artifact is also pushed into operational environment



Continuous Testing

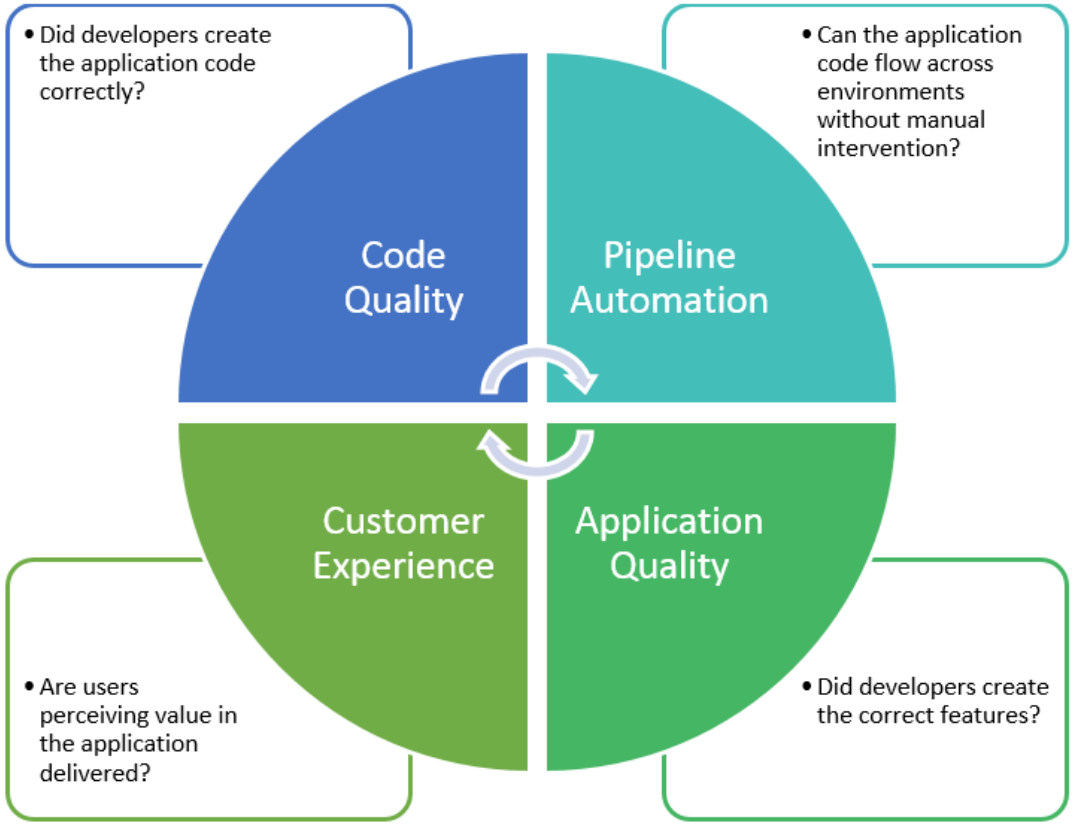
- Continuous Testing
  - Every artifact is tested as it is created
- Shift Left Model
  - Test early, test often
- CICD also adds
  - Automated testing at every stage
  - CT is triggered by events in the CICD process
  - Checking in code => automated unit testing
  - Build => integration testing





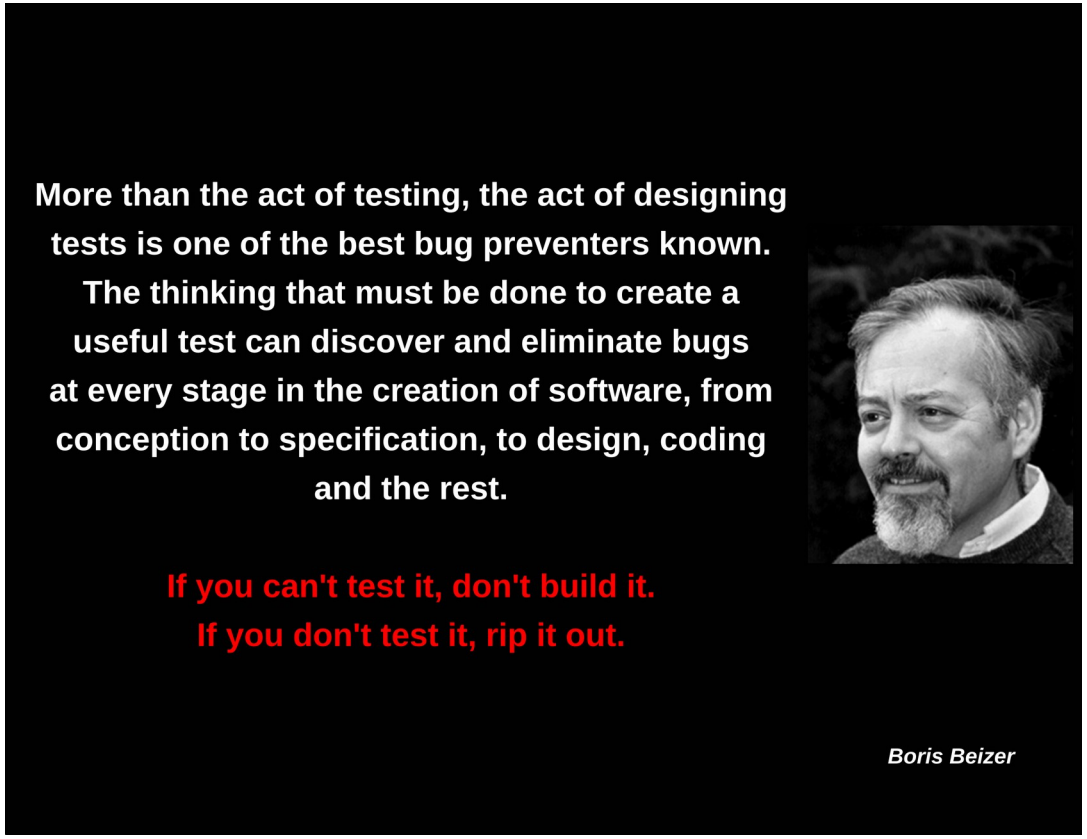
Continuous Testing

- Does not replace human based testing
  - Such as pair programming and code reviews or other similar processes
- Creates “quality gates”
  - Development pipelines abort when tests fail
  - Code has to pass tests in order to get to the next stage of a CICD pipeline
- Adding continuous security testing and security planning is called DevSecOps



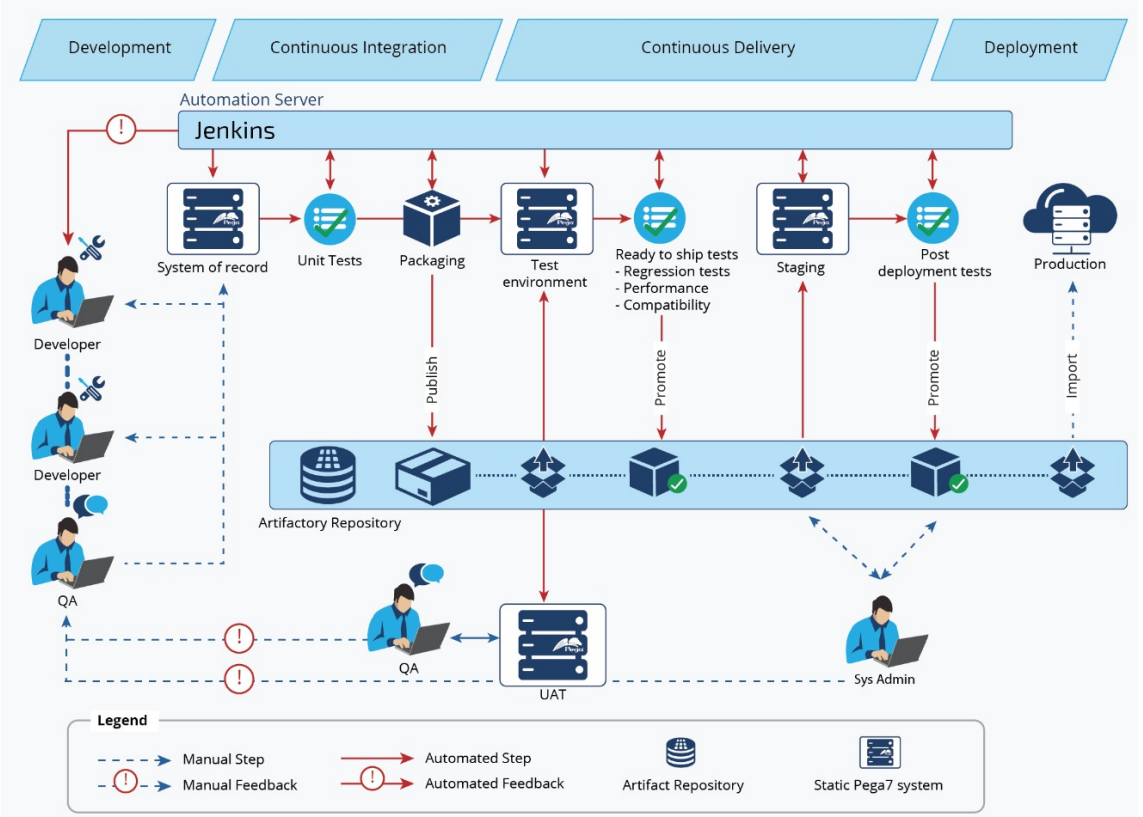
Continuous Testing

- Integrating testing into CICD
  - Forces a critical view of the specification
  - Identifies errors that might not be seen until later in the development when they are more expensive to rectify
- Especially errors of omission
  - Failure to identify systems behavior in cases of invalid or rare or unexpected inputs
  - If you can’t determine what the result of test should be based on the spec
  - Then you also don’t know what the code should do in that case



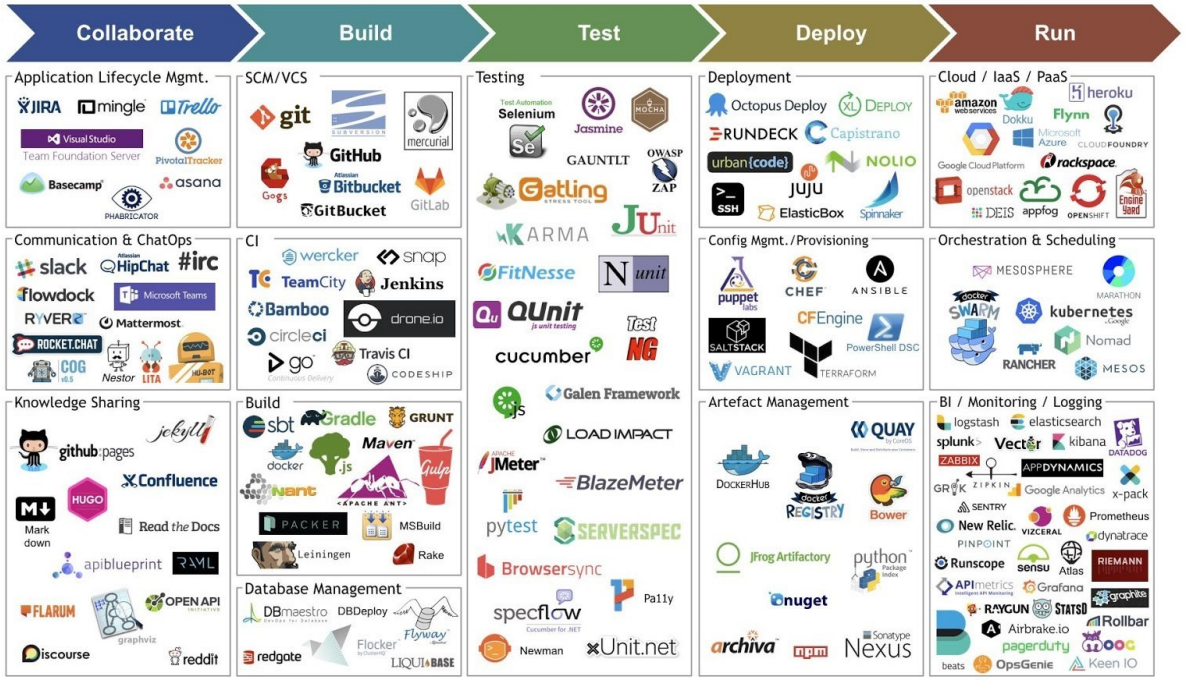
Pipelines

- Series of automated tasks
  - Implements a CICD flow
  - Managed by an orchestration tool
  - Jenkins is the tool used in the diagram
- The pipeline
  - Responds significant events like checking code in or automated tests all passing in automated integration testing
  - Responsible for moving the artifact to the next stage in the pipeline and initiating the activity of that stage



Automation Tools

- A wide range of CICD tools are available
  - Each automates some part of the pipeline
  - Lots of overlap
  - Not all are compatible
  - Wide range in quality
- Developing a toolset
  - Can be very problematic
  - Especially if some tools become obsolete



Automation Orchestration Tools

Product	Comment
Jenkins	Jenkins is the number one open-source project for CI/CD automation. Community publishes thousands of plugins to facilitate easier implementation of common tasks.
Travis CI	Used by companies such as Facebook, Mozilla, Twitter, Heroku,
TeamCity	TeamCity takes advantage of cloud computing by dynamically scaling out its build agents farm on Amazon EC2, Microsoft Azure, and VMware vSphere
Circle CI	CircleCI uses language-specific tools like rvm and virtualenv to ensure dependencies are installed into an isolated environment
CodeShip	Codeship is a hosted continuous integration platform that's focused on efficiency, simplicity, and speed. Teams can use Codeship to test, build, and deploy directly from projects on GitHub.
GitLab CI	GitLab CI provides tools for issue management, code views, continuous integration and deployment, all within a single dashboard.
Buddy	Custom support for Grunt. Gulp, MongoDB, and MySQL
Semaphore	Custom tests for dependencies, units, code style, security, and acceptance
AppVeyor	AppVeyor is a Windows-only cloud-based service for testing, building, and deploying Windows applications. SSD drives with dedicated hardware to provide fast speeds.
Google Cloud Build	Google's provider based solution for DevOps and CI/CD management
AWS CodePipeline	Amazon Web Services's provider based solution for DevOps and CI/CD management
Azure DevOps	Microsoft Azure's provider based solution for DevOps and CI/CD management

CICD Benefits

- Smaller code changes are simpler (more atomic) and have fewer unintended consequences
- Fault isolation is simpler and quicker
- Mean time to resolution (MTTR) is shorter because of the smaller code changes and quicker fault isolation
- Testability improves due to smaller, specific changes. These smaller changes allow more accurate positive and negative tests
- Elapsed time to detect and correct production escapes is shorter with a faster rate of release
- The backlog of non-critical defects is lower because defects are often fixed before other feature pressures arise
- The product improves rapidly through fast feature introduction and fast turn-around on feature changes
- Upgrades introduce smaller units of change and are less disruptive

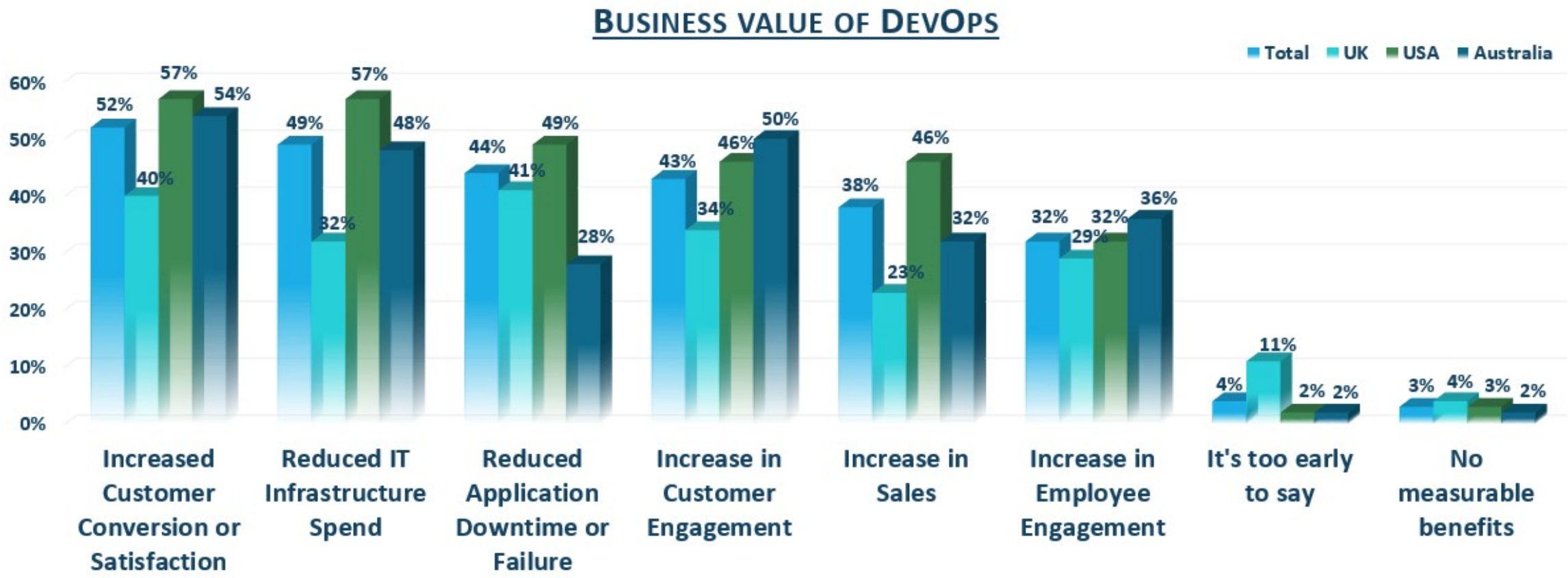
CICD Benefits

- CI-CD product feature velocity is high. The high velocity improves the time spent investigating and patching defects.
- Feature toggles and blue-green deploys enable seamless, targeted introduction of new production features.
- You can introduce critical changes during non-critical (regional) hours. This non-critical hour change introduction limits the potential impact of a deployment problem.
- Release cycles are shorter with targeted releases and this blocks fewer features that aren't ready for release.
  - End-user involvement and feedback during continuous development leads to usability improvements. You can add new requirements based on customer's needs on a daily basis.

https://help.mypurecloud.com/articles/benefits-continuous-integration-continuous-deployment-ci-cd/

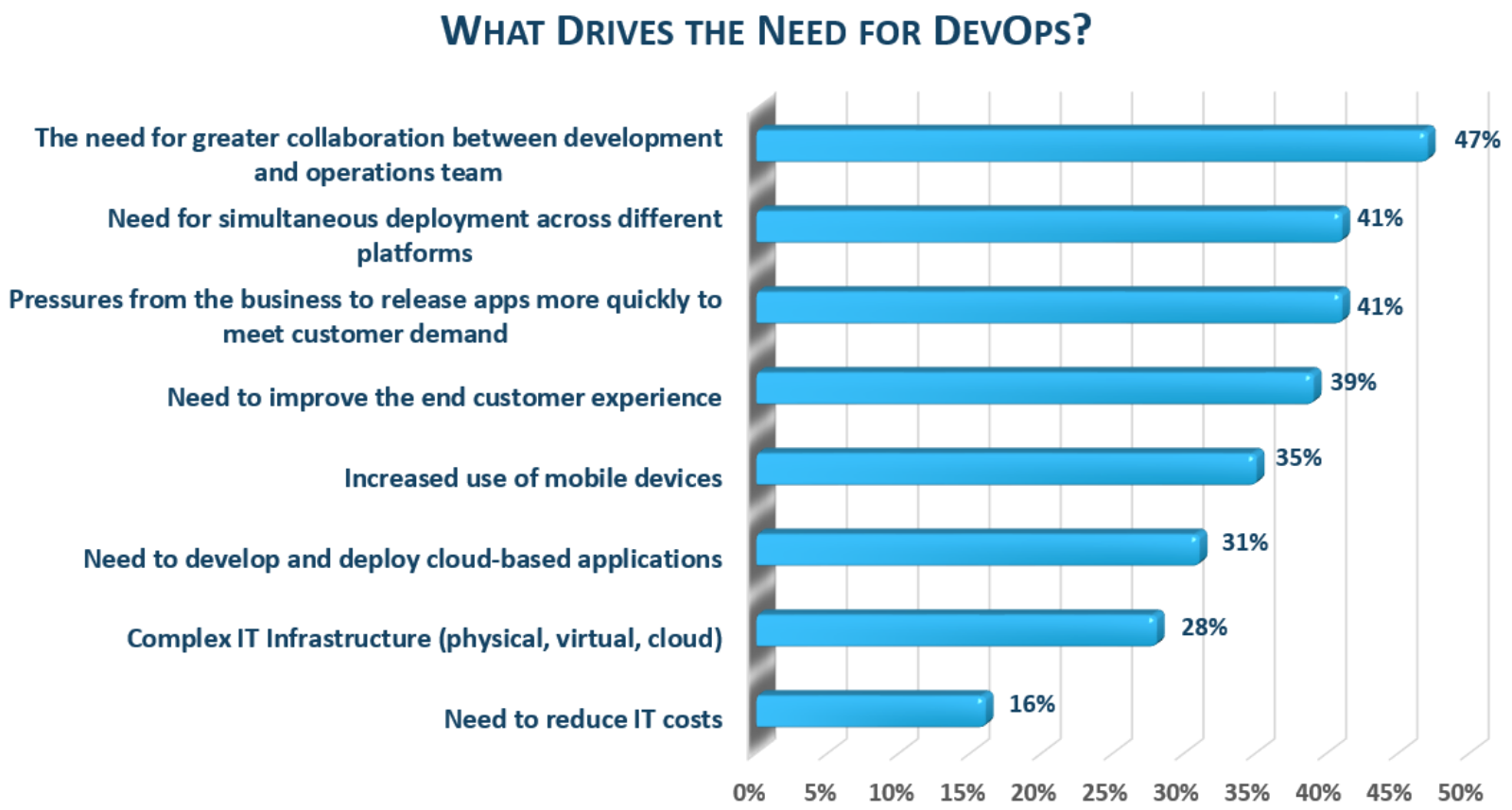
CICD Benefits

The chart actually focuses on CICD but considers it part of DevOps. The productivity gains are primarily due to the CICD practices



CICD Drivers

The chart actually focuses on CICD but considers it part of DevOps.



IBM Internal DevOps

Functions	Previous Time Frame	Present Time Frame	DevOps Benefit
Project initiation	10 days	2 days	80% faster
Overall time to development	55 days	3 days	94% faster
Build verification test availability	18 hours	< 1 hour	94% faster
Overall time to production	3 days	2 days	33% faster
Time between releases	12 months	3 months	75% faster

DevOps, clearly an extension of lean and agile principles, was as much, in IBM, born of necessity to respond to a pervasive industry mandate to "do more with less" and has evolved to "quality software faster."

- Kristof Kloeckner,
General Manager,
IBM Software Group – Rational

https://www.compaid.co.in/Articles/DevOps-Value-and-Cost-Savings

Challenges for CICD

- Organization silos and corporate culture
  - Lack of communication between development, QA and operations
- Failure to automate testing or do continuous testing
  - QA starts lagging behind development requiring rework to fix buggy code
- Legacy systems integration
  - Automated tools may not be available for legacy systems
  - E.g. Unit testing frameworks for COBOL code
- Complexity and size of applications
  - Trying to apply CICD to too big a “chunk” of development
  - Especially when introducing CICD improvements

CI and CD in Agile Development

- CICD are essential ingredients for teams doing iterative and incremental software delivery in Agile Development
  - Developers share a common source code repository
  - Dedicated Continuous Integration environment
  - All code must pass unit tests
  - Integrate often
  - Regression tests run often
  - Code metrics are published
  - Every change to the system is releasable to production
- Automation is the key
  - The process is too time consuming and tedious to do manually

Continuous Integration (CI)

- CI is where members of a team integrate their work frequently
  - usually each person integrates at least daily, leading to multiple integrations per day
- Each integration is verified by an automated build (including test)
  - Intended to detect integration errors as quickly as possible
  - Goal is to merge and test the code continuously to catch issues early by automating integration process
- A CI project must have a reliable, repeatable, and automated build process involving no human intervention
  - CI Server (orchestration tool) is responsible for performing the integration tasks
- Automatic unit testing, static analysis and failing fast are core to CI

Continuous Integration Practices

- Single source repository for all developers
- Build automation
  - Every change to the integration branch should trigger a new build
- Keep the builds fast and trackable
  - Make the builds self-testing
- Test the builds in production-like environment
  - Keep all verified releases in artifacts repository and available to everyone
- Publish coding metrics

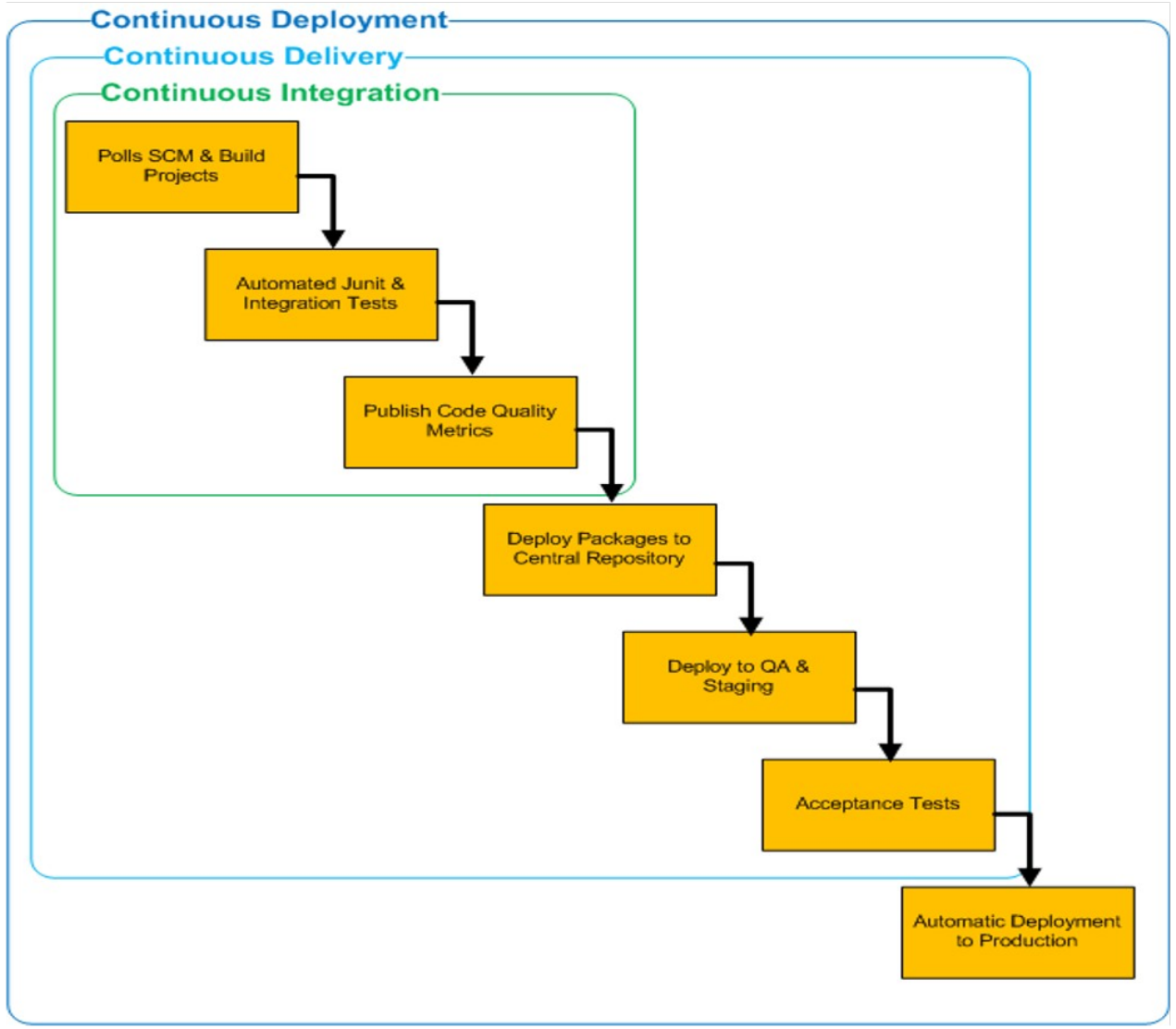
Continuous Delivery (CD)

- CD is a natural extension of CI
  - Every change to the system has passed all the relevant automated tests and is ready to deploy in production
  - Team can release any version at the push of a button
  - Keep all verified releases in artifacts repository and available to everyone
- But the deployment to production is not automatic
  - The goal of CD is to put business owners in the control of scheduling of the software releases
  - The decision to release is a governance decision, not a technical one
-

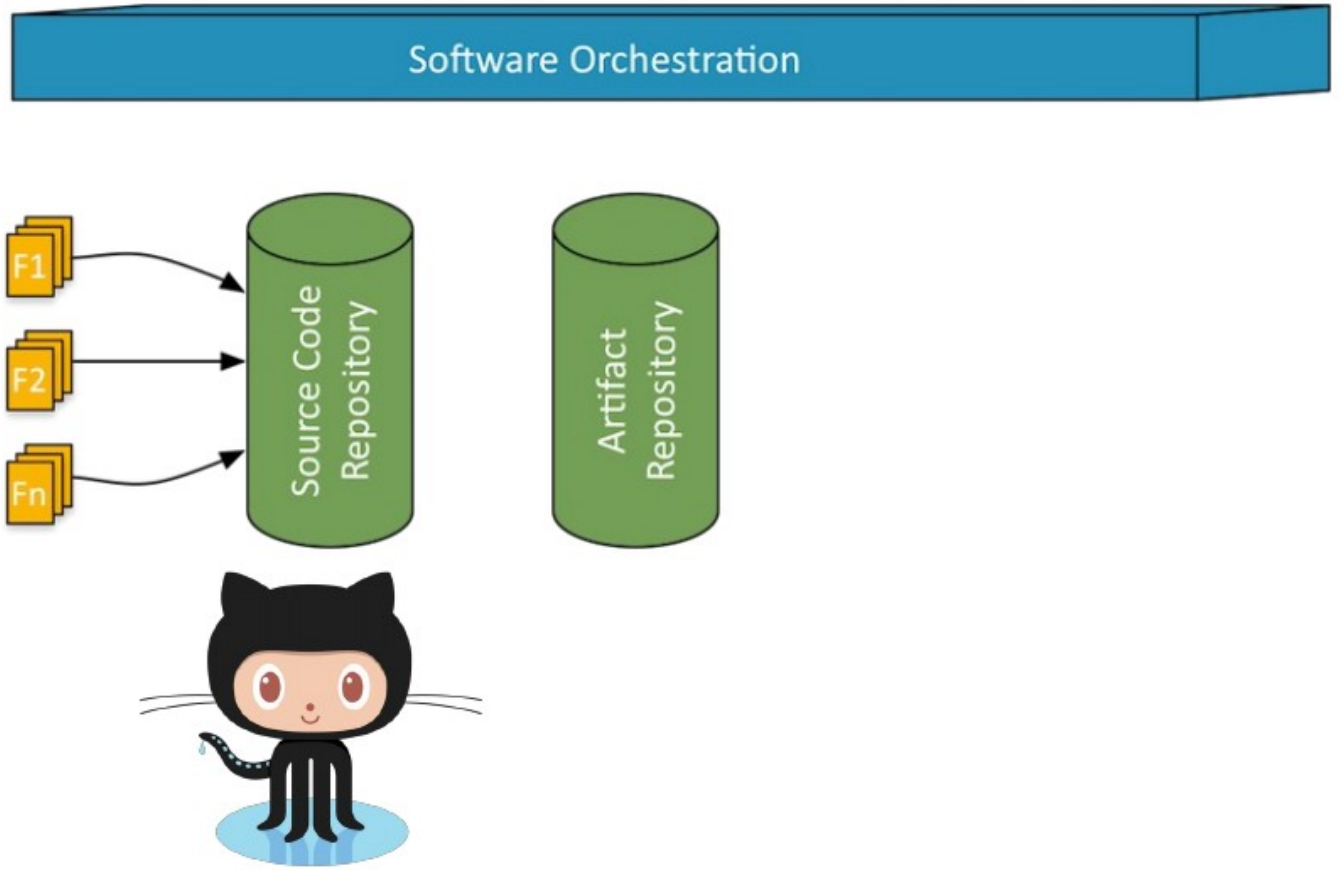
Continuous Deployment (also CD)

- Continuous Deployment adds automatic deployment to end users in the Continuous Delivery process
  - Continuous Deployment automatically deploys every successful build directly into production
  - Deploying the build to production as soon as it passes the automated and UAT tests
- Continuous Deployment is not appropriate for many business scenarios
  - Business Owners prefer more predictable release cycles as opposed to arbitrary deployments
  -

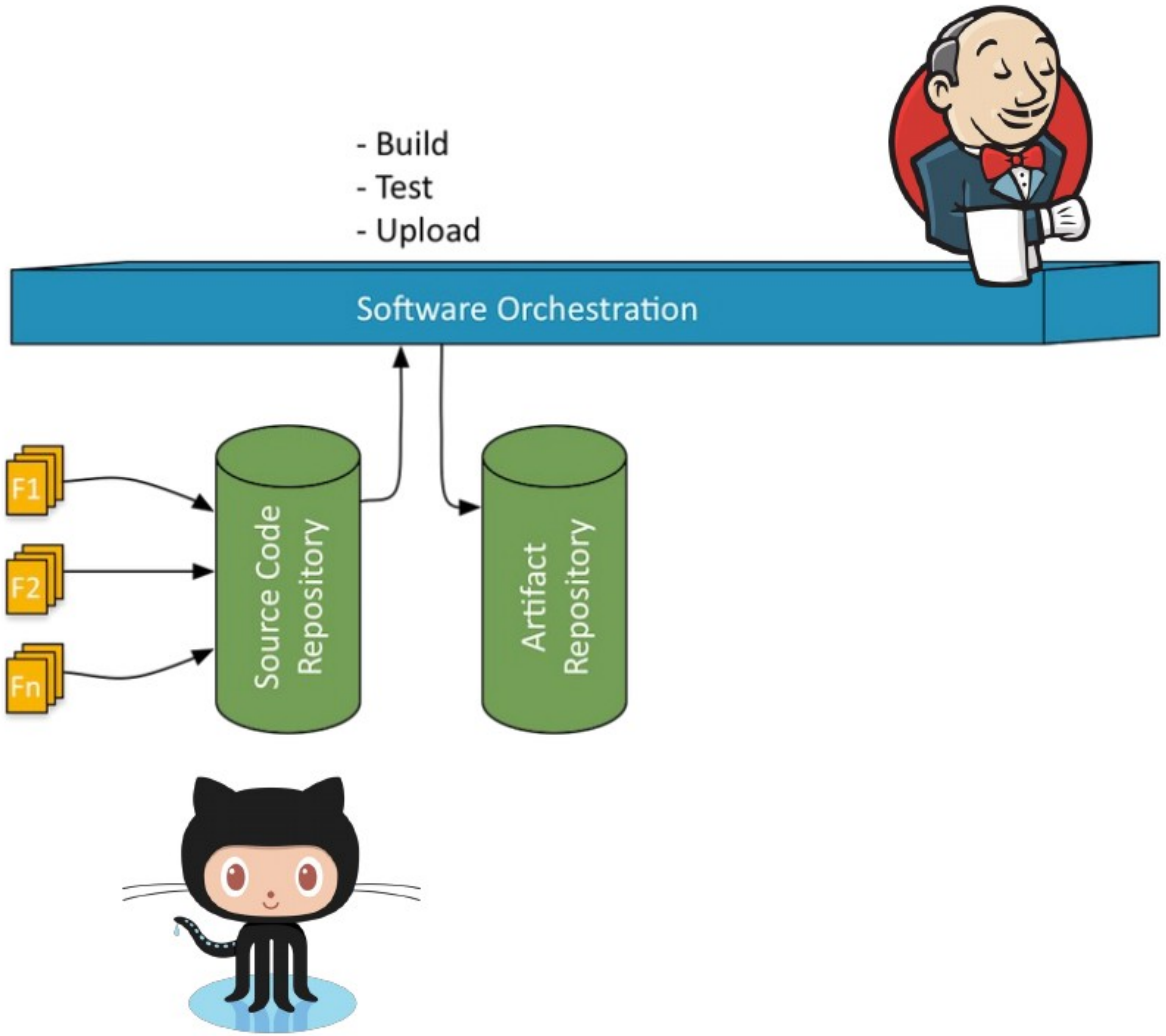
Continuous Integration, Delivery and Deployment



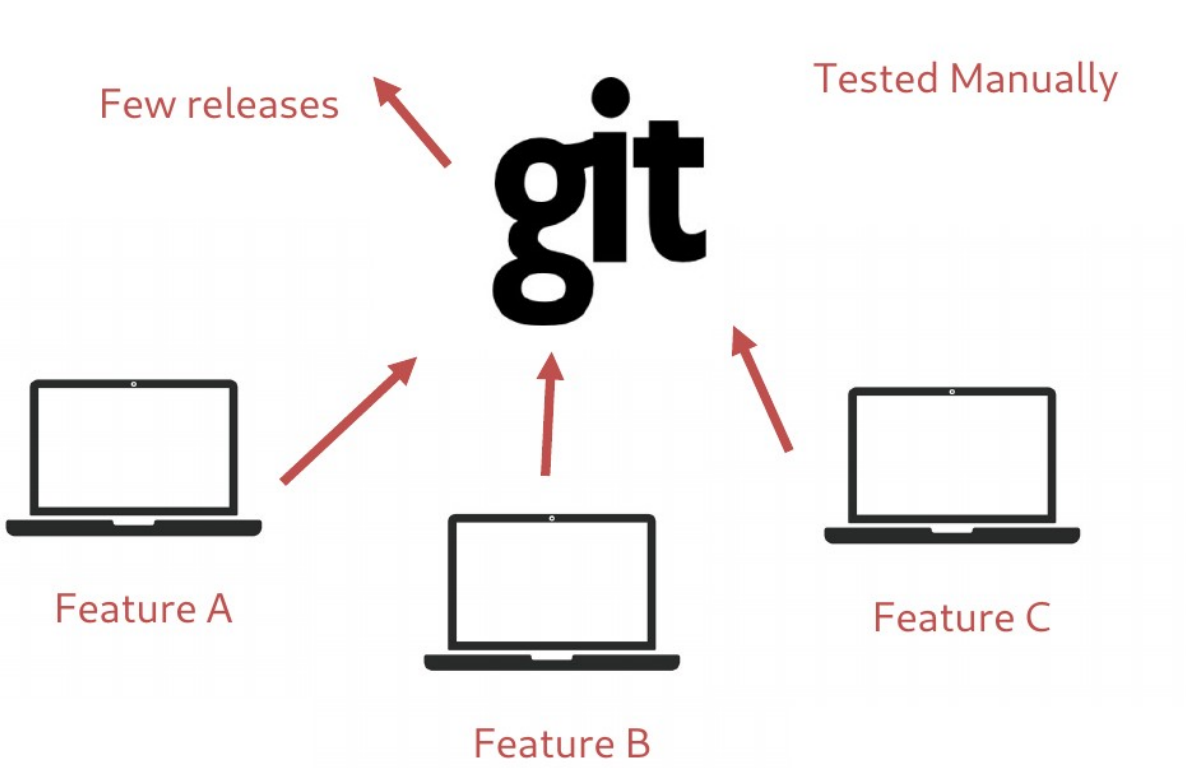
Continuous Integration



Continuous Integration

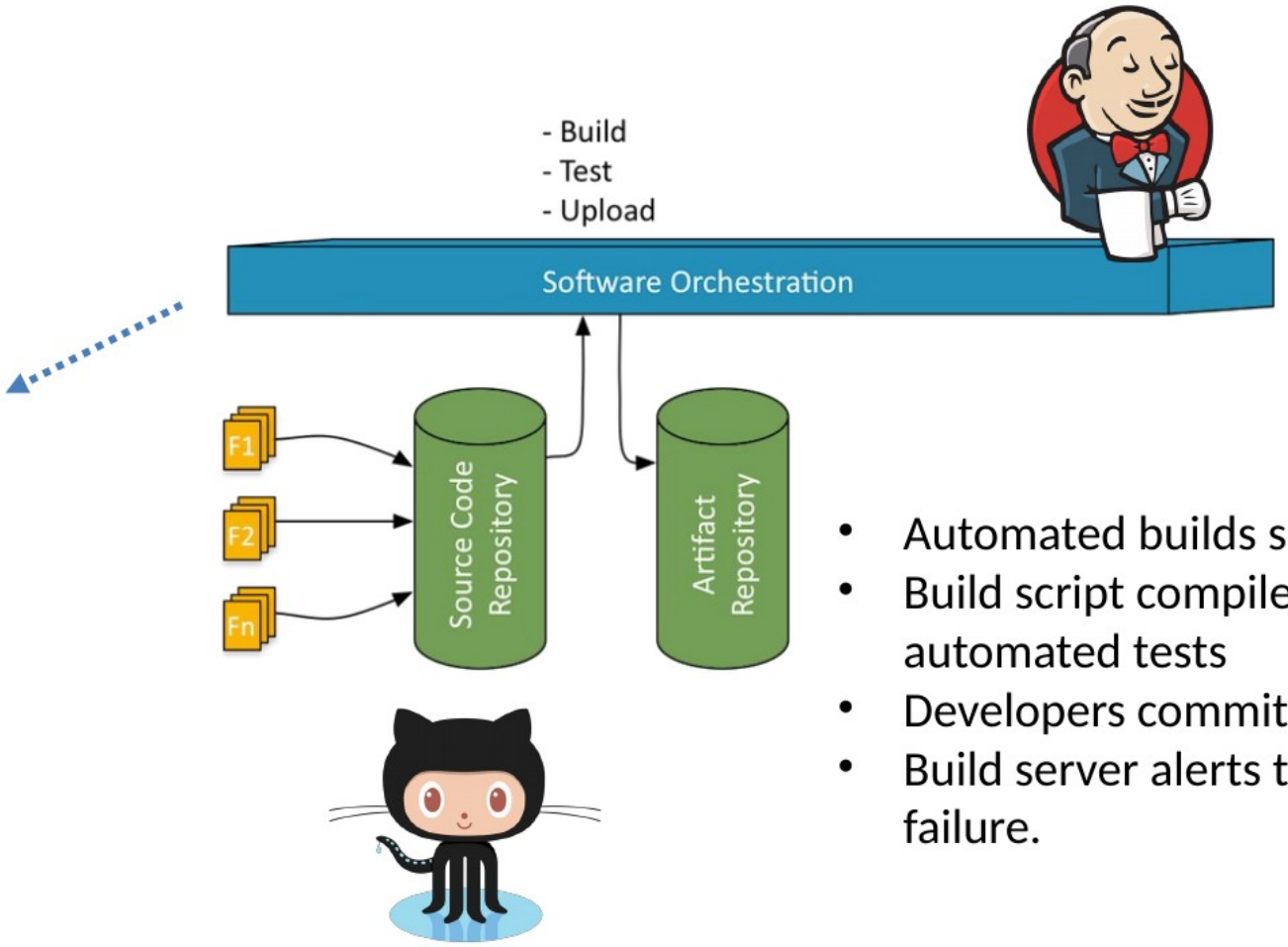


CI Adoption



- No build servers
- Developers commit on a regular basis
- Changes integrated/tested manually
- Fewer releases

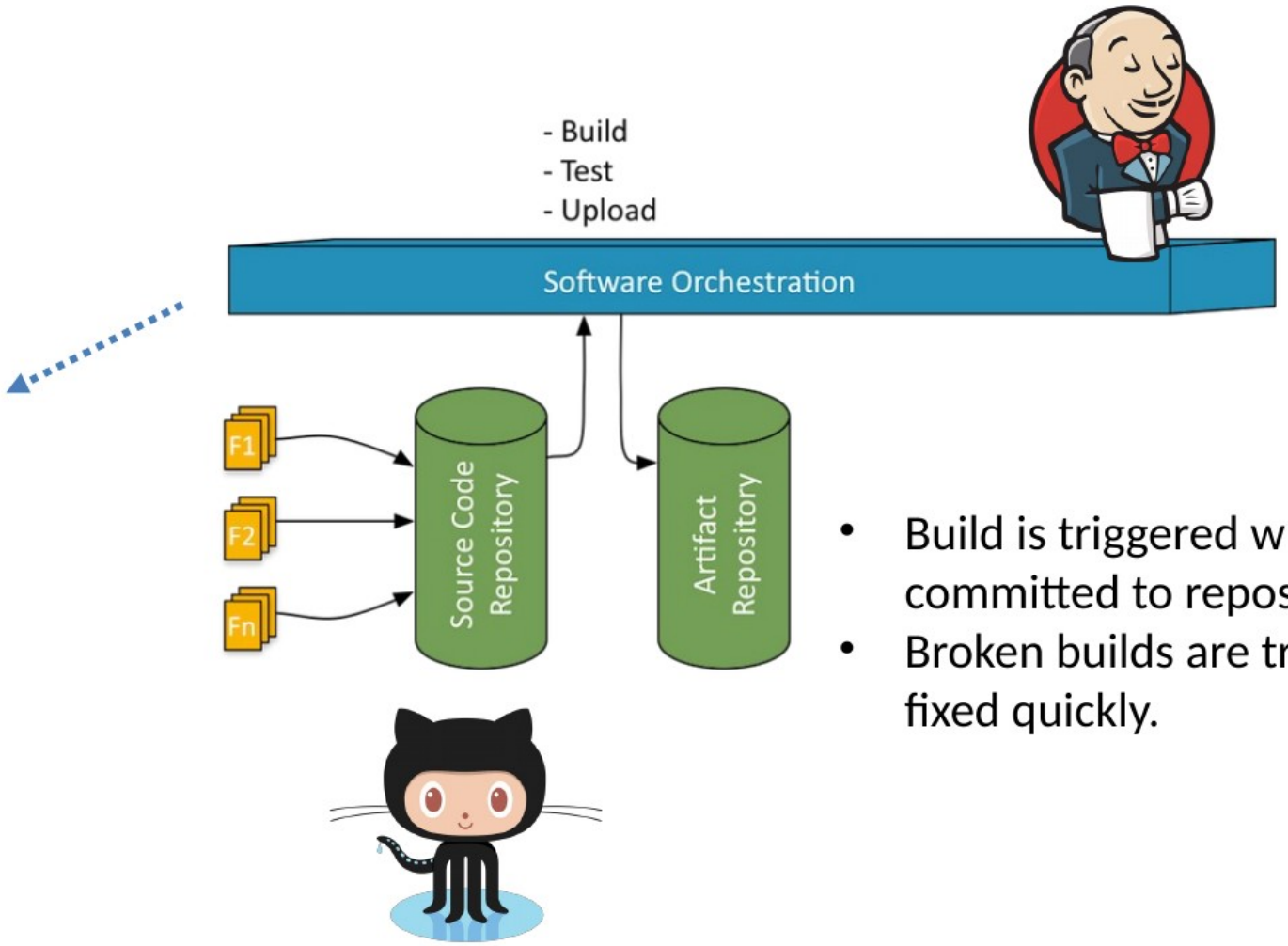
CI Adoption



Nightly builds

- Automated builds scheduled regularly
- Build script compiles app and runs set of automated tests
- Developers commit changes frequently
- Build server alerts team members on build failure.

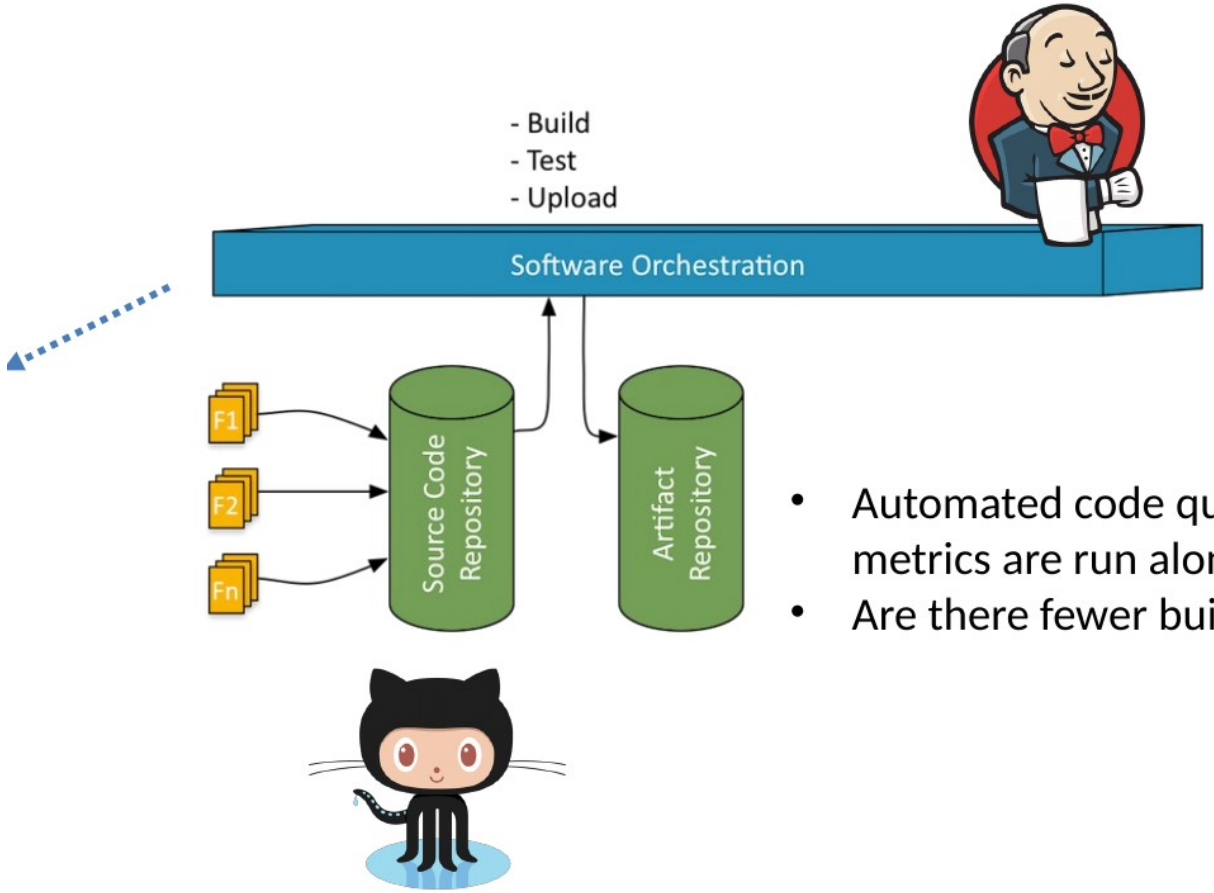
CI Adoption



Triggered automatically

- Build is triggered whenever new code is committed to repository.
- Broken builds are treated as high priority and fixed quickly.

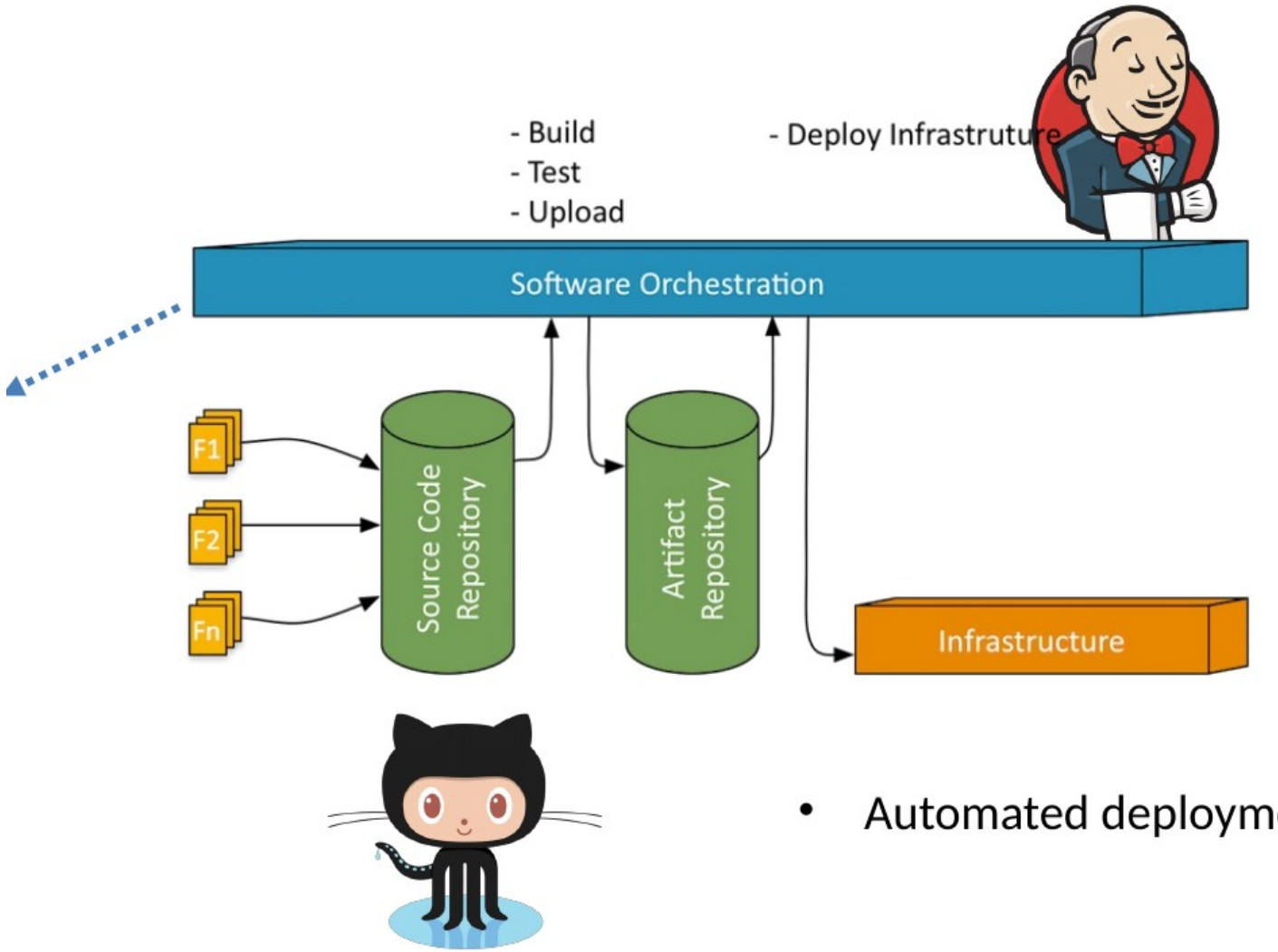
CI Adoption



Triggered automatically

- Automated code quality and code coverage metrics are run along with unit tests.
- Are there fewer build failures?

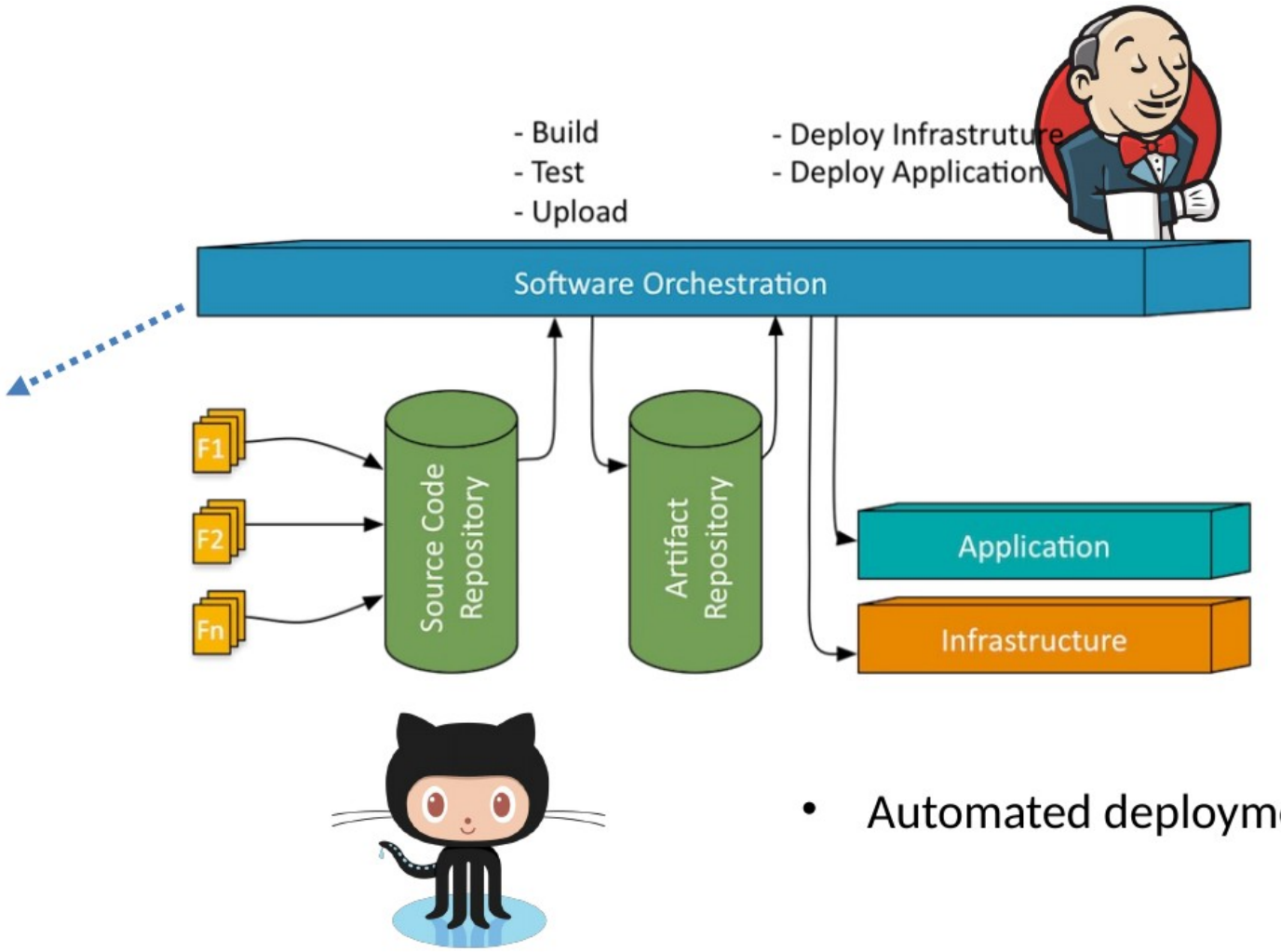
CI Adoption



Triggered automatically

- Automated deployment capabilities

CI Adoption



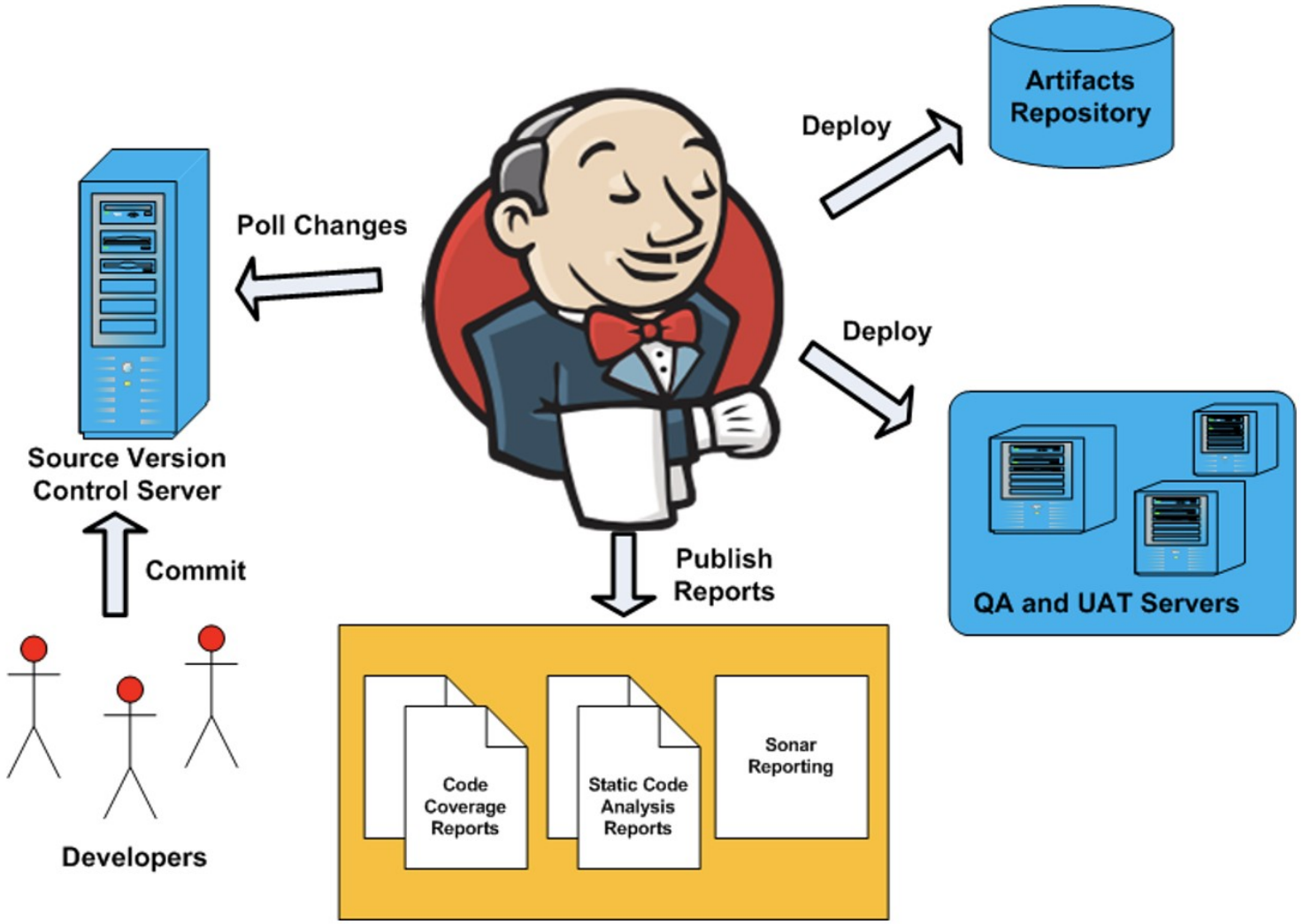
Triggered automatically

- Automated deployment capabilities

Jenkins

- Continuous Integration (CI) Server
  - Open Source, free and written in Java
  - Large and dynamic community with massive adoption
  - Easy to install on many different platforms
  - Easy to use with a user-friendly interface
  - Significant amounts of resources and tutorials available
  - More than 1000 plugins
  - New plugins released every week
  - Extensible architecture – easy to extend and customize
  - Distributed builds

How Jenkins Fits in CI and CD



Jenkins

- Jenkins is derived from Hudson
  - Hudson was first released by Kohsuke Kawaguchi of Sun Microsystems in 2005
  - Initially it was only used within Sun but by 2010 Hudson captured 70% of CI market share
  - Oracle bought Sun Microsystems in 2010
  - Due to naming and an open-source dispute, original Hudson team forked Jenkins from Hudson as a new project
  - Oracle continued the development of original Hudson
  - Majority of Hudson users migrated to Jenkins within few months of the initial Jenkins release

Managing Jenkins

- All in one configuration dashboard
  - Configure JDKs, Ant, Maven, Security, Email and Version Controls
  - Install new plugins and update any existing plugins
  - Configure parallel and distributed builds
- View Jenkins logs and statistics in real time

Jenkins Distributed Architecture

- The main node is the default node
  - Runs the Jenkins instance
  - Manages access to Jenkins Interface
  - Should be the only node accessible via web interface
- The master/main node is the controller
  - Best practice is that CICD jobs are delegated to other nodes
- Agent/Slave nodes
  - Tasked with running parts of the CICD jobs
  - Agents are tagged so they can be used selectively
  - Jenkins does not have to be installed on an agent
  - Just Java to run the Jenkins agent jar file
  - Communication takes place via ssh

Jenkins Pipelines

- A pipeline defines a CICD process for that project
- Stages
  - Sequential series of steps to be executed, for example
  - Build > Test > Package > Deploy
- JenkinsFile (infrastructure as code)
  - A script that defines the stages of a pipeline
  - Older form is written in the Groovy scripting language
  - New form is a declarative form of scripting
- Three basic configurations
  - The JenkinsFile is kept in Jenkins
  - The JenkinsFile is kept in a SCM repository
  - The JenkinsFile is kept in the project itself
    - This allows for multi branch builds

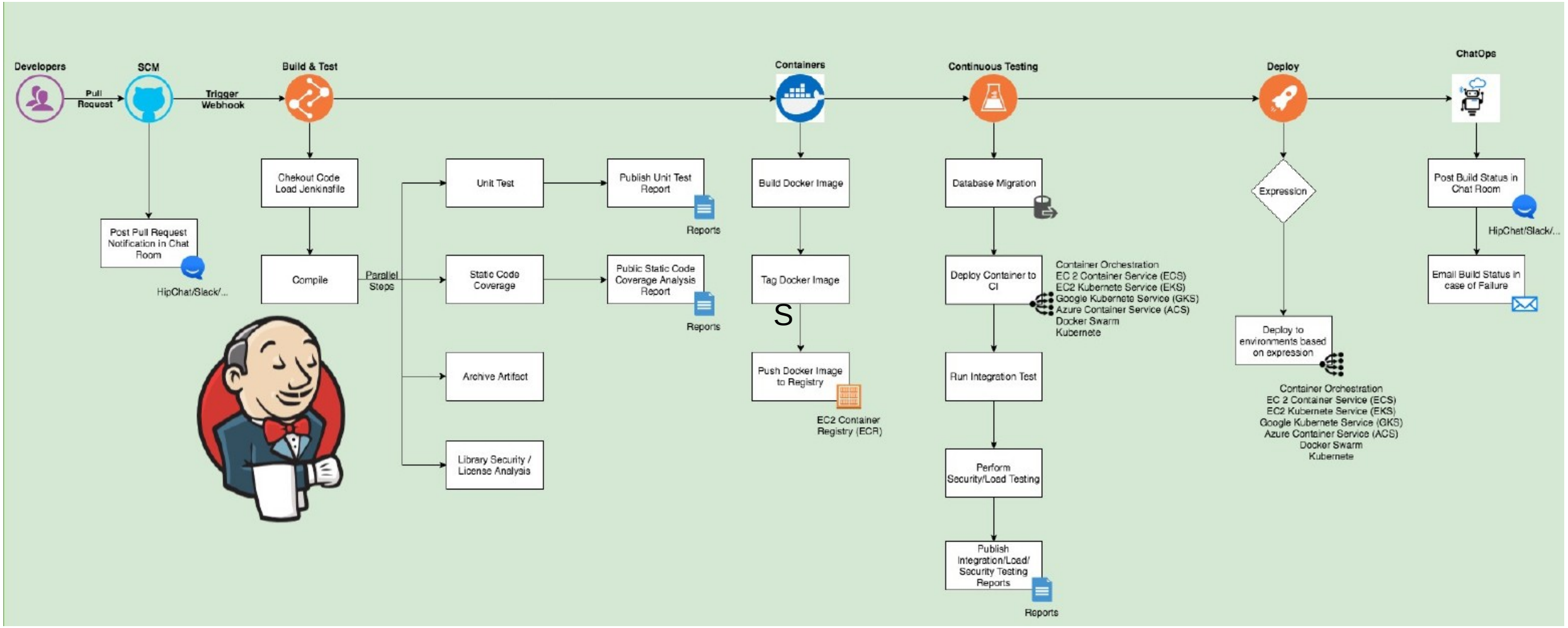
Post Build Steps

- In addition to stages, Jenkins has a post build stage
- Contains any of a number of clause types
  - Always – always executes
  - Failure – executes only when the pipeline fails
  - Success – executes only when the build succeeds
  - Cleanup – always runs after all the other clauses run
  - Changed – runs if the pipeline or a stage completion status is different from a previous run
- There are more clause types at:
  - https://jenkins.io/doc/book/pipeline/syntax/#post

Environment Variables

- There are a number of predefined environment variables
  - Accessed via the global variable “env”
  - Value is accessed using Groovy syntax
    - “Build ID is ${BUILD_ID}”
  - We can define environment variables in either the whole pipeline or a given stage using the environment block

Pipeline


Developers
SCM
Pull Request
Trigger Webhook
Build & Test
Containers
Continuous Testing
Deploy
ChatOps
Post Pull Request Notification in Chat Room
HipChat/Slack/...
Chekout Code Load Jenkinsfile
Compile
Parallel Steps
Unit Test
Static Code Coverage
Archive Artifact
Library Security / License Analysis
Publish Unit Test Report
Reports
Public Static Code Coverage Analysis Report
Reports
Build Docker Image
Tag Docker Image
Push Docker Image to Registry
EC2 Container Registry (ECR)
Database Migration
Deploy Container to CI
Container Orchestration
EC 2 Container Service (ECS)
EC2 Kubernete Service (EKS)
Google Kubernete Service (GKS)
Azure Container Service (ACS)
Docker Swarm
Kubernete
Run Integration Test
Perform Security/Load Testing
Publish Integration/Load/ Security Testing Reports
Reports
Expression
Deploy to environments based on expression
Container Orchestration
EC 2 Container Service (ECS)
EC2 Kubernete Service (EKS)
Google Kubernete Service (GKS)
Azure Container Service (ACS)
Docker Swarm
Kubernete
Post Build Status in Chat Room
HipChat/Slack/...
Email Build Status in case of Failure

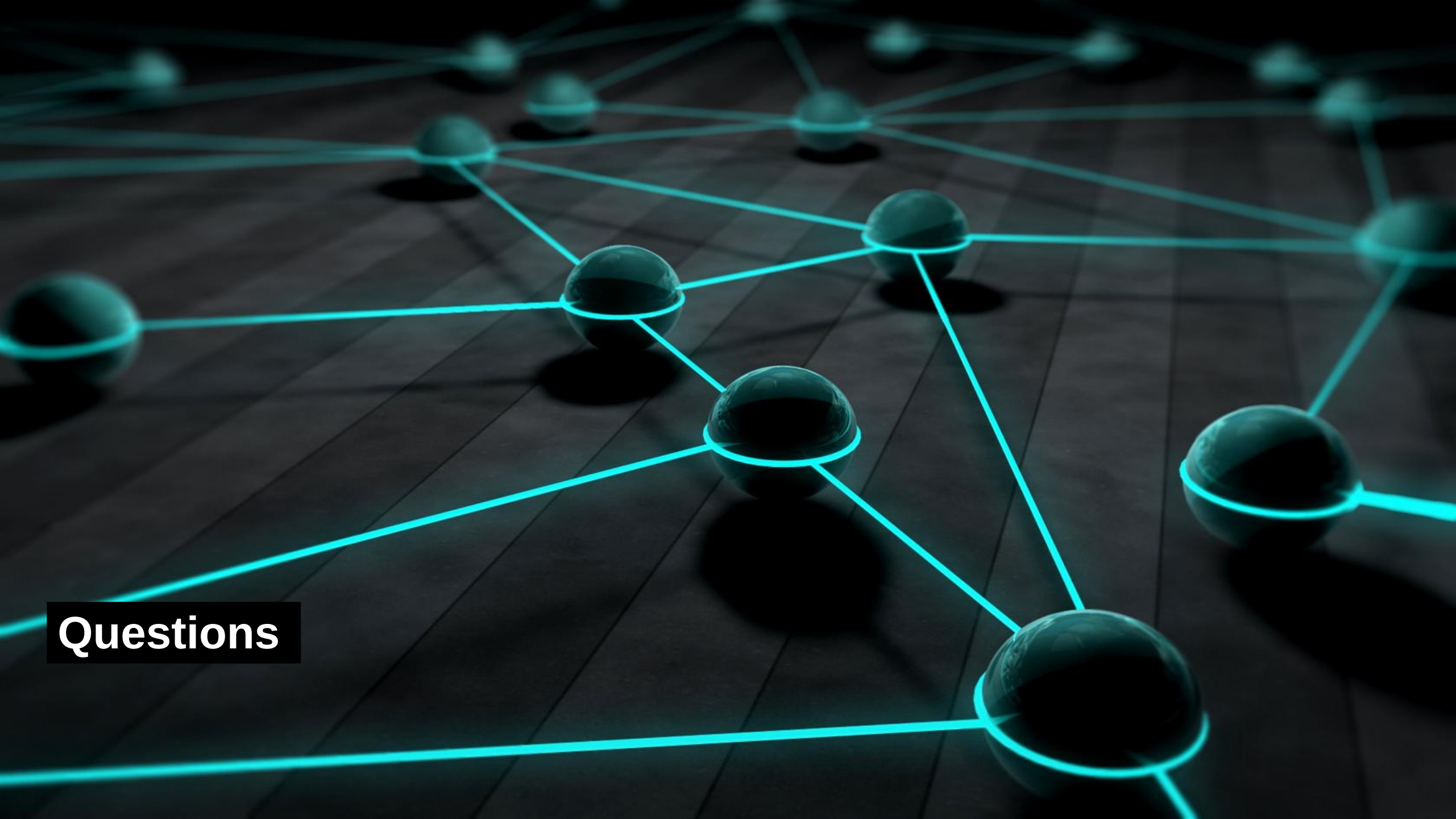
Pipeline Features

- Can support complex, real-world, CD Pipeline requirements
  - Pipelines can fork/join, loop, parallel, and more
- Resilient:
  - Pipeline executions can survive master restarts
- Can be paused:
  - Pipelines can pause and wait for human input/approval.
- Pipeline configuration is treated as as code in source control

Pipeline Stage View

billing-rest - Stage View

	Declarative: Checkout SCM	Initialize	Checkout	Build	Publish Reports	SonarQube analysis	ArchiveArtifact	Docker Tag & Push	Deploy - CI	Deploy - QA	Deploy - UAT	Deploy - Production	Declarative: Post Actions
Average stage times: (Average full run time: ~2min 59s)	768ms	1s	799ms	57s	5s	14s	125ms	23s	37ms	32ms	32ms	32ms	792ms
#118 Jul 17 20:58 — 1 commit	817ms	1s	690ms	1min 36s	10s	24s	198ms	38s	38ms				1s
#117 Jul 16 15:28 — 1 commit	792ms	1s	708ms	1min 36s	8s	23s	179ms	38s	42ms				2s
#116 Jul 15 21:13 — No Changes	672ms	869ms	694ms	1min 33s	10s	24s	183ms	37s	40ms				75ms


Questions