Deloitte.

DevOps Playbook

Click on a section to start

If this is your first visit, please navigate to how to use the playbook

How to use the playbook

Contact & Submit Feedback

DevOps Essentials
Nutshell
Value Stream

DevOps in a Nutshell

A way of thinking and acting that **builds on Agile** and **Lean Thinking** to bring additional **speed to** deliver technology with greater **stability**, **quality** and **security**.

Continuous Integration

CI

Build, unit test, integrate daily
- Automate build, unit test
- Integrate source code with technical quality
- Metrics to manage technical quality

Automated Test

AT

Automate functional, stress testing
- Frequent regression testing
- Resolve defects early
- Metrics to manage functional quality

Continuous Delivery

CD

Test deployment daily
- Automate deployments and releases
- Run deployment scripts and deployment validation tests
- Resolve issues early

- Breaking down the barrier between Dev and Ops
- Driving manual touch out of IT processes through automation
- Applying a "continuous everything and shift left" mindset to every IT process

Resulting in…

- Dramatic increase in speed to market
- Improved quality and operational stability
- Optimized value and productivity of scarce and expensive human labor

Automate Infrastructure Provisioning
- Build complete infrastructure from source code repository
- Tight control over all computing environment configurations

SDI

Software Defined Infrastructure

Foundation for DevOps
- A merge of "dev" and "ops" best practices
- A capability to strengthen when starting DevOps

CM

Configuration Management

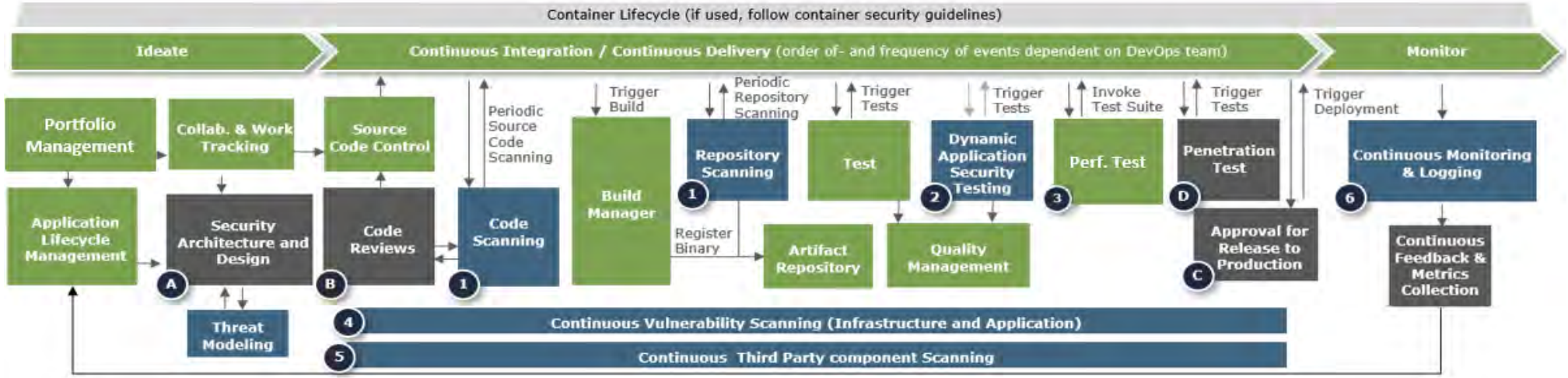
Automate Service Management
- Orchestrate and automate the end-to-end event response process
- Leveraging APIs for handoffs

ZTO

Zero Touch Operations

CYBER SECURITY

DevOps Value Stream with inbuilt security practices



Color Legend

Green	DevOps Processes
Blue	Security Automation
Gray	Security Processes

Built-in Security Automation

The following are commonly identified security gaps in a DevOps Pipeline:

1. Static code scanning to support code quality and secure coding standards (e.g., Fortify, SonarQube, Veracode, Checkmarx)
2. Dynamic application security testing for applications is performed (e.g., WebInspect)
3. Automates performance / load testing for all web applications are completed (e.g., VSTS)
4. Continuous vulnerability scanning of all infrastructure and applications (e.g., Nexus, Qualys)
5. Continuous third party component scanning for vulnerabilities (e.g., Sonatype)
6. Continuous logging and monitoring of production environment (e.g., Splunk)

Manual Security Processes

Checkpoints and gates must be implemented so security controls are in place:

- **A** Security and architecture designs are reviewed and documented in a central repository (e.g., Confluence, etc.)
- **B** Code reviews are enabled in the workflow and secure coding standards are upheld (e.g., VSTS Pull Requests)
- **C** Gates/checkpoints are in place to ensure that release to production goes through the appropriate approval process and exceptions are managed
- **D** Periodic penetration tests are performed (e.g., OWASP Zap)

What is Version Control?

Version control or source code control or revision control systems are designed to allow organizations to maintain a complete history of their application changes across all artifacts. They enable teams to work on separate parts of an application and yet maintain a source of record.

Distributed Version Control System (DVCS)

- Each user keeps a self-contained, first-class repo on their computer and can work offline
- Keeps a complete history of file changes in the repo, making it fast and easy to swap between versions
- Scalable, highly available and fault-tolerant
- Flexible to support many workflows and branching models

Artifacts Managed

- Application Source Code
- Test Cases and Suites
- Configuration Files
- Schema Generation Scripts
- Pipeline/Automation Code
- File used to create, compose container orchestration
- **NOT Binaries and Images**

Benefits

- Providing visibility to all stakeholders
- Reverting to a point in history
- Tracing and auditing
- Enabling parallel development
- Corelating changes

Git Overview, Commands and Workflows

What is Git?

Git is the most widely used open source distributed version control system (DVCS) created by Linus Trovalds. It's been forked to create many commercial DVCS solutions (eg. GitLab, GitHub).

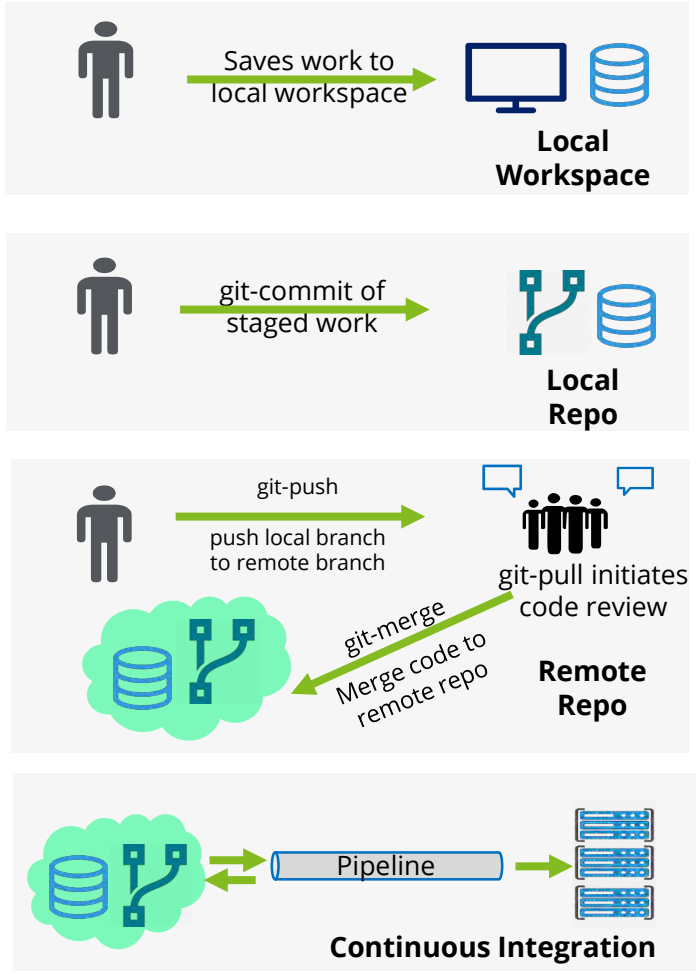
How it works?

- **Local Workspace** is where developers do their work
- **Local Repo** is your version of the remote repo where your code gets committed with a "git-commit" command or through git extensions or IDE plugins
- **Remote Repo** is where your changes gets pushed, pulled or merged for other team members to see

Common Git Operations

- git-commit : Commits work to local repo
- git-rebase : Pulls in changes from one branch to another, usually from trunk to feature branch
- git-pull : Request for review and merge changes to remote repo for other team members to see
- git-cherry-pick : Choose a specific commit from one branch and apply to another, usually from trunk to Release branch

Typical Developer Workflow



1. Developer completes the code changes and saves them in his local IDE workspace

2. Developer stages all his changes and executes the '`git-commit`' command to commit the changes to his local repo

3. Developer pushed local branch to remote branch. He creates a pull request '`git-pull`' to tag reviewers for the latest commit to be merged/applied to an upstream branch. Once approved, 'git-merge' applies changes to the remote repo

4. Once the new changes are applied, the CI/CD pipeline gets kicked off.

To Branch or not to Branch?

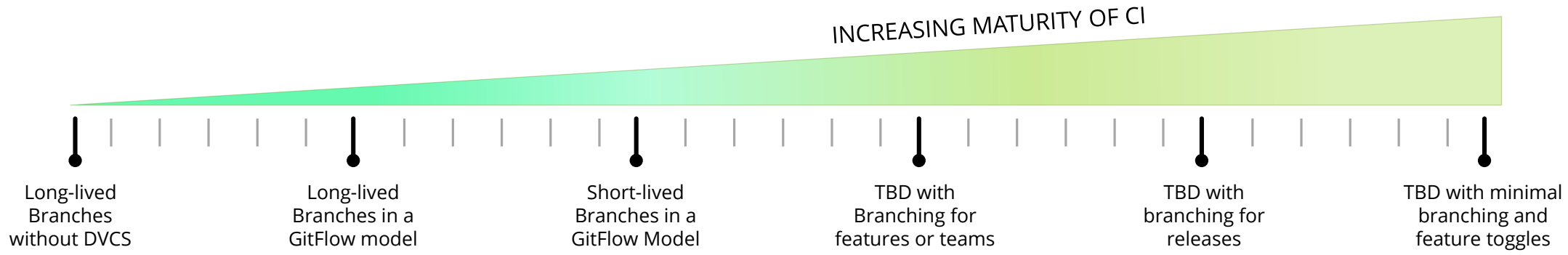
Branching creates a copy of your trunk or another branch to enable parallel development of multiple workstreams. **Branching, at its core, is antithetical to the principles of Continuous Integration**. Every branching decision is trade-off against Continuous Integration.

GitFlow

- Branching as the core strategy, multiple long-lived branches
- Some common types of branching
  - Functional (Patches, release, products)
  - Physical (Files, components, Subsystems)
  - Environmental (Platforms, operating systems, hardware)
- Unless for releases, changes from one branch will almost always have to be **merged**/applied to another

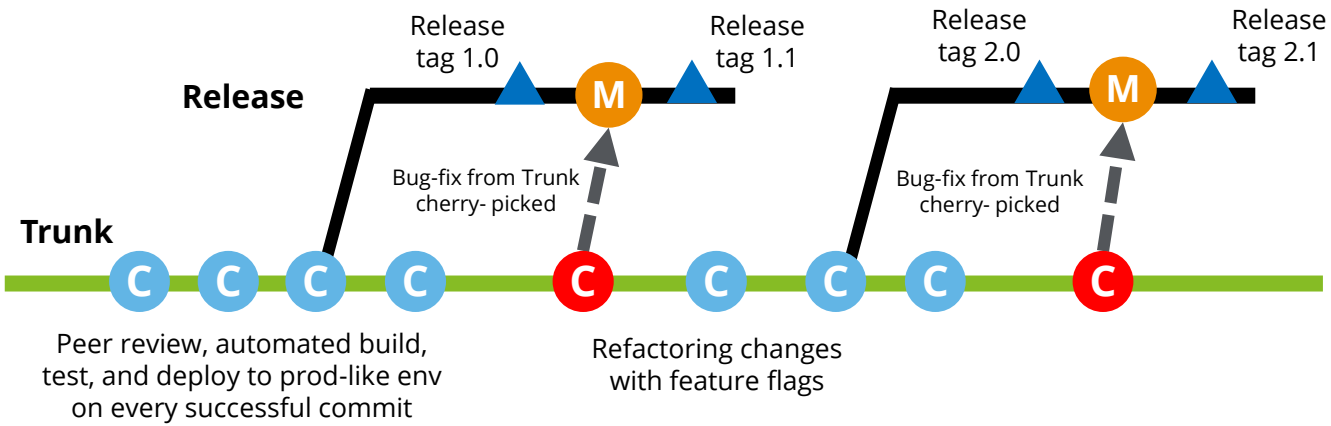
Trunk-based Development (TBD)

- Developers **always check-in to trunk** and branching is rarely used, mostly for releases and not merged back to the trunk
- Trunk is **always deployable** with a CI/CD pipeline
- TBD with multiple teams and large systems/platforms requires **good componentization**, incremental development, and **feature toggling and abstraction**



TBD with Branching Only for Releases

The one situation when it's always acceptable to create branches is shortly before a release. Development should continue on trunk with production defect fixes cherry picked to the release branch.

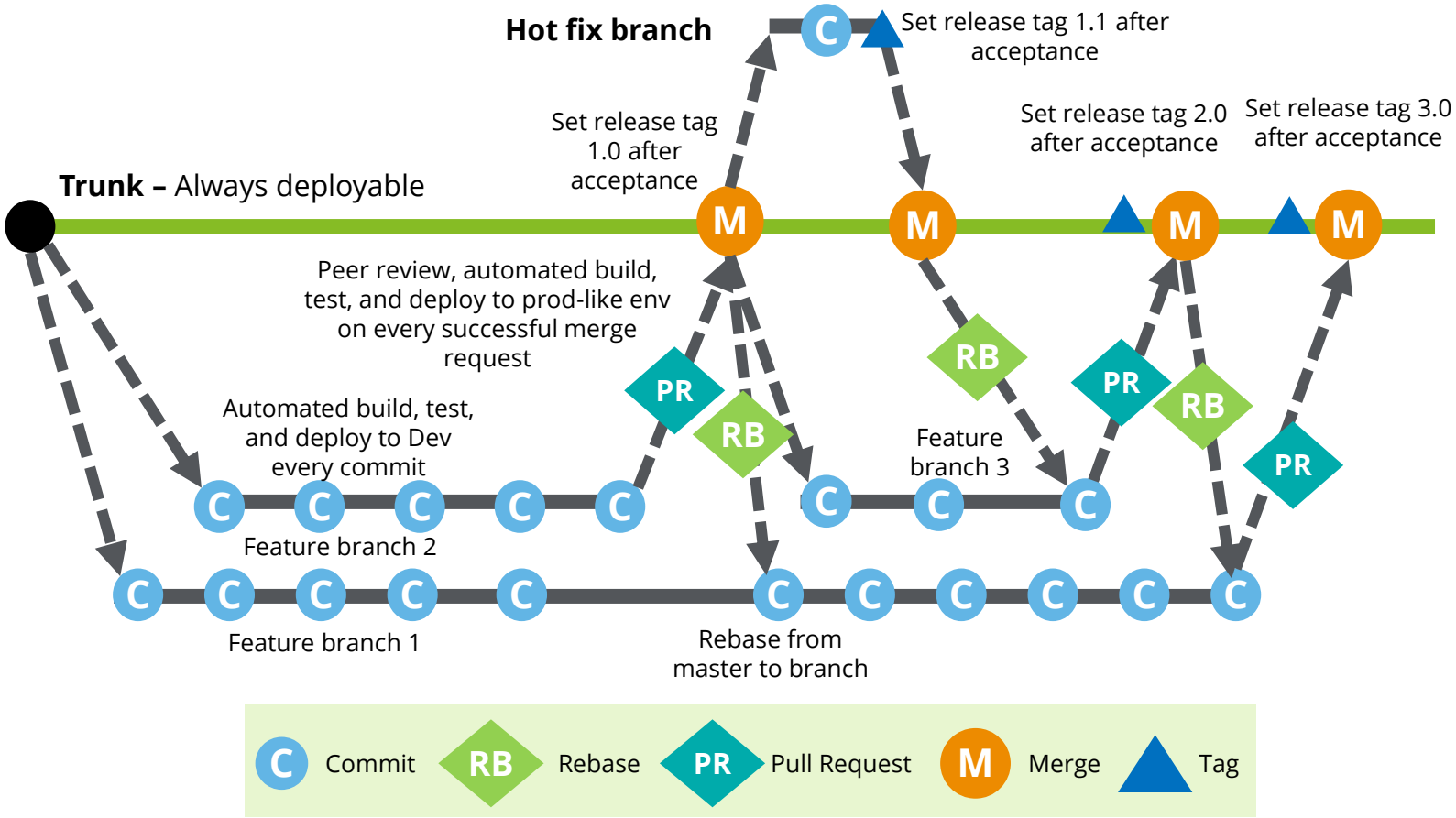


Quick Nuggets

- Branch only when necessary, with Release branches as snapshots, and delete after new release goes in.
- Rebase during local commit and merge frequently
- Bugs should be fixed on trunk, and then *cherry-picked* to the release branch
- Very large platform teams sometimes create release branches for each component, merged later to an *integration branch*
- *Branch by Abstraction* or *feature flags* for large refactoring projects
- Open-source projects like kubernetes use this model with robot merging from branches to Trunk
- Only Developers or DevOps Engineers with Release duties should check-in to the Release branch

TBD & Short-Lived Branches (GitHub Flow)

This model is the most widely used DVCS model as it gives teams the flexibility of working concurrently in feature branches for a few days and yet keep the Trunk in a deployable state. It takes discipline to not let Trunk being undeployable and feature branches becoming long-lived.

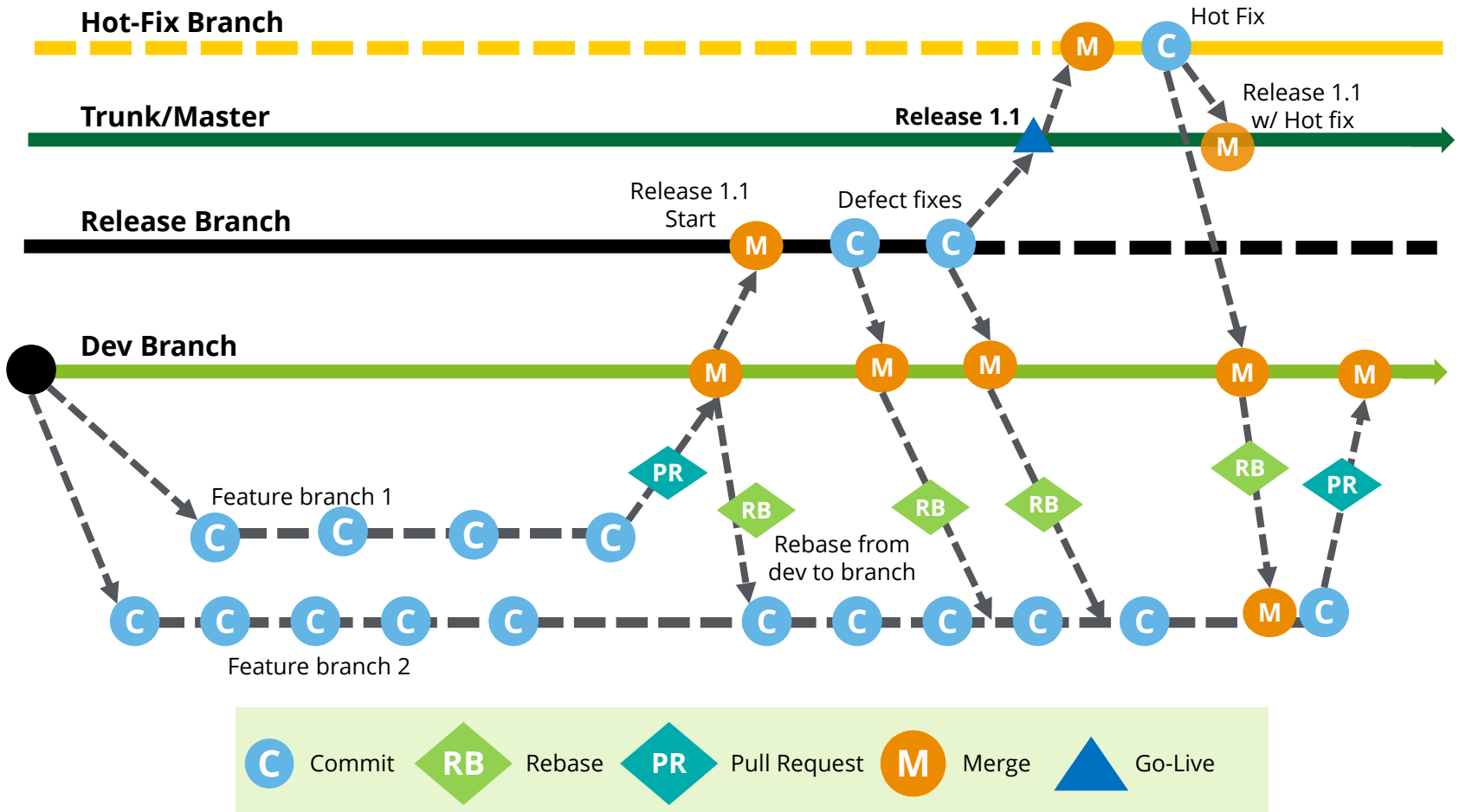


Quick Nuggets

- Branches should be short lived, mapped to user stories, and deleted after merge
- Rebase during local commit and merge frequently
- Only 1-2 developers per branch
- Use multi-branch features to create automatic Pipelines for short-lived Feature branches
- Conduct code-reviews with pull requests before merging to Trunk

GitFlow Branching Model

GitFlow is incompatible with Trunk-Based Development and has been the preferred model in a lot of enterprises, with heavy/centralized release processes and lack of componentization. It's trade-off between isolating and breaking changes from the Trunk for additional process.



Quick Nuggets

- Branching limited to only two long-lived branches – Trunk and Dev
- Integration in Dev, hardening in Release and deploy to PROD from Trunk
- Merge frequently to Dev and keep feature branches short-lived
- Only 1-2 developers per branch feature branches
- NEVER branch off an existing branch to create a "staircase"
- Only Developers or DevOps Engineers with Release duties should check-in to the Release branch

Wrapping security around version control & development

Left-shift security requirements to provide a more effective and cost efficient approach by allowing for security to be designed in versus remediated at the end of the project lifecycle.

Plan

Establish integrated operations and baseline security capabilities

Development

- Approved code base
- Security service inventory
- Joint story repository / backlog
- Embedded security architecture review
- Threat modeling

DevOps Toolchain

- Pipeline architecture & management
- Access controls
- Source code management

Infrastructure

- Architecture management
- Data protection/masking
- Redundancy
- Access controls (e.g. IAM integration)
- Container / VM hardened baseline

Quick Nuggets

- Perform overarching security architecture review
- Define user risk stories to address security requirements based on policies and guidelines
- Create and update application-specific threat modeling

DevOps Processes
- Version Control
  - Basics
  - Leading Practices
  - Tools
- Continuous Integration
- Continuous Testing
- Continuous Deployment
- Continuous Monitoring

Version Control – Lead Practices

Core Capabilities

#	Leading Practice	Key Considerations	Key Benefits
1	App Code and Static Content with Full Version History	▪ All the application code and static content (e.g., CSS, HTML) should be in the source code management(SCM) tool (e.g., Git, Subversion, CVS) that is used to track changes to software projects. ▪ Version control systems should allow developers to automatically track their work, see a history of all changes, and revert to previous versions of a project when needed.	▪ Allows developers to work simultaneously on the code ▪ Allows developers to isolate their changes, if required, through branching
2	DB Scripts used for Schema Creation and Data Migration	▪ Ensure that version control acts as a single source of truth by ensuring that all database code (DDL, procedures, reference keys, access grants) is covered. ▪ All scripts must be “environment aware”, and only relevant changes should be deployed. ▪ DB changes must be tied with the user stories for traceability purpose.	▪ Supports DB deployments within the agile sprint schedule ▪ Improves repeatability and execution frequency of the database deployment process
3	Automated Tests and Manual Test Scripts	▪ Version control should include everything needed to test a software/service. This includes test script (e.g., unit, acceptance, stress) and test data. ▪ Either the test data or source data/SQL scripts that generate the test data can be checked in.	▪ Easily revert to the older version of test scripts ▪ Improves repeatability of testing process with ability to create all pre-requisite data for the execution
4	Code Build, Packaging, and Deployment Scripts	▪ Practicing continuous delivery (CD) requires automation and management of the build, packaging, and deployment scripts, which can be addressed through version control check-ins. ▪ Version control should act as a single source of truth and allow easy traceability/rollback in the event of build/deployment issues.	▪ Improves repeatability of the process ▪ Accelerates onboarding of new team members
5*	All Check-ins Traceable to Requirement or Defect Work Items	▪ Avoid the trap of creating traceability documentation (e.g., traceability matrix, lifecycle traceability) that requires constant maintenance. ▪ Leverage features of tools like JIRA and version control systems (e.g., smart commits, commit hooks) to automate the traceability between code check-ins and stories. ▪ Focus on implementing the required level of traceability driven by domain complexity, geographic distribution of the team, and regulatory requirements.	▪ Easily identify the affected part of the code if tests fail ▪ Identify and eliminate redundant test cases

*** Definitions are provided in the appendix.**

Version Control – Lead Practices

Advanced Capabilities

#	Leading Practice	Key Considerations	Key Benefits
6*	Environment Configuration and Provisioning Scripts	▪ Create foundation for infrastructure as code to allow automatic management and provisioning of IT infrastructure without manual intervention and in a consistent way regardless of the environment. ▪ Allow implementation of a code review policy for the provisioning scripts. Any changes should be submitted as a pull request and be approved by a peer reviewer/SME before being accepted.	▪ Improves the consistency of the infrastructure setup ▪ Allows enforcement of code review policies ▪ Allows environment configuration testing using tools similar to those used to test code ▪ Allows the use of static analysis and code coverage tools on configuration scripts
7	Trunk-Based Development, Branch for Release	▪ Developers should collaborate on code in a single branch (called 'trunk') instead of creating other long-lived branches, thereby avoiding merge and build issues. ▪ Trunk-based development is a key enabler of continuous integration (CI) and, by extension, continuous delivery (CD). ▪ Individuals on a team should commit their changes to the trunk multiple times a day to satisfy the core CI requirement that all team members commit to trunk at least once every 24 hours. ▪ Depending on the intended release cadence, release branches may be cut from the trunk on a just-in-time basis and 'hardened' before a release. Those branches are deleted some time after the release.	▪ Ensures that the codebase is always releasable on demand and helps to make CD a reality ▪ Allows scale up/down of the development team with no impact on quality/throughput ▪ Eliminates merge issues prevalent with branching

*** Definitions are provided in the Additional Resources.**

Version Control – Lead Practices

Security Capabilities

#	Leading Practice	Key Considerations	Key Benefits
8	Repository scanningfor credentials, PII	• Scans repositories for secrets, keys and credentials. The entire commit history and all branches are scanned. This is done through either scanning for keywords or calculating Shannon entropy of strings.	▪ Helps to catch any secrets accidentally committed

*** Definitions are provided in the appendix.**

DevOps Processes
- Version Control
  - Basics
  - Leading Practices
  - Tools
- Continuous Integration
- Continuous Testing
- Continuous Deployment
- Continuous Monitoring

Development – Lead Practices

Core Capabilities

#	Leading Practice	Key Considerations	Key Benefits
1	Coding Standards and Self-Documenting Code	▪ Code written with understandable naming and structured programming conventions can eliminate the need to maintain comments or separate documentation. ▪ Consistent coding standards (e.g., strong cohesion, decoupling, portability, unit tests) can improve code readability and maintainability.	▪ Accelerates learning curve and reduces the risks involved in updating legacy code ▪ Reduces the need for developers to consult secondary documentation such as code comments or software manuals that tend to get out of sync with the code
2	Able to Build and Unit Test Locally	▪ Builds and unit tests should be run on developer's local machine and the build server. ▪ Unit tests should be fast and stateless and have limited external dependencies. ▪ Developers' local workspace should be set up with required libraries (with external dependencies stubbed/mocked) to build code and should preview it prior to checking in to source control.	▪ Provides instant feedback to the developer if the new/modified code is building and working as expected ▪ Supports adoption of trunk-based development by ensuring that the code check-ins do not break anything
3	Unit Test Automation with Stubbing/Mocking and 90%+ Code Coverage	▪ The success of agile delivery relies on the ability to maintain the codebase in a deployable state at all times. Hence, the unit test suite should be run as part of the build process. ▪ **State Verification:** Determine whether the exercised method worked correctly by examining the state of the subject under test (SUT) and its collaborators after the method was exercised. ▪ **Behavior Verification (Mocking):** Determine whether the exercised method worked correctly by examining the calls made by the SUT, telling the mock what to expect during setup, and asking the mock to verify itself during verification.	▪ Unit test execution as part of CI builds facilitates earlier detection of any issues with the new code checked-in ▪ Allows verification of the functionality of a given method in isolation from other calls (e.g., database)
4	Rigorous Refactoring	▪ Refactoring should occur during code fixes. Time should also be allocated as part of sprint capacity to reduce technical debt through code refactoring. ▪ Unit test execution ensures that the system is kept fully working after each small refactoring, reducing the chances that a system seriously breaks during the restructuring.	▪ Monolithic routines are deconstructed into a set of concise, well-named, single-purpose methods ▪ Easier to modify and extend the application capabilities through the use of recognizable design patterns
5	Shared Codebase and Collective Code Ownership	▪ Maintain collective code ownership to abandon any notion of individual ownership of modules. ▪ The code base should be owned by the entire team; anyone may make changes anywhere. ▪ The entire team is accountable for keeping the codebase in a buildable/deployable state at all times.	▪ Reduces the amount of time spent waiting for other people to make required code changes to address dependencies ▪ Encourages collective responsibility at a team level

DevOps Processes
- Version Control
  - Basics
  - Leading Practices
  - Tools
- Continuous Integration
- Continuous Testing
- Continuous Deployment
- Continuous Monitoring

Development – Lead Practices

Advanced Capabilities

#	Leading Practice	Key Considerations	Key Benefits
6*	Pair Programming	▪ Employ pair programming to improve skills when team members have diverse capabilities and skillsets. ▪ Employ a "test first" mentality. Developers can pair up in a test-driven development (TDD) model; one developer writes the test, another writes/updates the code, then the first developer can refactor the code.	▪ Improves transfer of skills; junior developers can learn from more experienced team members ▪ Eliminates the need for a separate code review exercise ▪ Improves overall code quality/maintainability ▪ Reduces coordination efforts; N/2 pairs to coordinate instead of N individual developers
7*	Test-Driven Development	▪ Avoid dependencies between test cases to provide the flexibility of running only a subset. ▪ Organize test cases to reflect organization of the application code.	▪ Achieves user-oriented design consisting of highly cohesive, loosely coupled components ▪ Leads to modular, extensible, and flexible code ▪ Reduces technical debt through code cleanup
8*	Behavior-Driven Development	▪ Extend TDD methodology with participation from business/stakeholders in terms of defining the "desired behavior" more accurately.	▪ Establishes a closer relationship to acceptance criteria for a given function and the tests used to validate that functionality
9*	Microservices Architecture	▪ Microservice architecture should support agile development by using cross-functional teams. Each team is responsible for making specific products based on one or more individual services communicating via message bus.	▪ Improves fault isolation because other services remain largely unaffected by the failure of a single service ▪ Eliminates lock-in to a single technology stack for entire application/services ▪ Reduces dependency concerns associated with a monolithic application

*** Definitions are provided in the Additional Resources.**

Development – Lead Practices

Security Capabilities

#	Leading Practice	Key Considerations	Key Benefits
10	Security architecture design	▪ Integrate security in architecture and design to reduce an application's attack surface and to embed appropriate controls before development starts; includes manual security architecture review and threat modeling exercises ▪ Involve information security personnel in the design of application/services and software demos during development. ▪ Security review needs to be part of the development cycle, not a separate process that occurs post development. Development teams have explicit security requirements for their features. ▪ Pre-approved libraries and tools in shared repositories for all development teams. For example, recommended configurations for application/service frameworks (e.g., 2FA, Bcrypt, Logging) and management of secrets (keys, settings, and processes) are accessible and well documented. ▪ Developers can run local security scan before committing the code. Role-based authentication is implemented across SDLC tools.	▪ Improves access control mechanism and prevents un-authorized access to privileged data ▪ Builds credibility ▪ Enables security to be an inherent part of the architecture and hence the application
11	Threatmodeling	▪ Threat modeling is a process by which potential threats, such as structural vulnerabilities can be identified, prioritized, and avoided, using a hypothetical attacker's point of view	
11	Static Code Analysis and Automated Security Reviews	▪ All new and existing code should be analyzed by a code analysis tool to provide an understanding of the code structure and to help ensure that the code adheres to the security/industry standards. ▪ Automated tools can help programmers and developers conduct the static analysis as part of the continuous integration (CI) process. ▪ As part of code change process, developers should use the IDE code analysis plug-ins to identify and address quality issues.	▪ Reveals errors that do not manifest themselves until a disaster occurs; promotes a comprehensive, software quality control regime ▪ Creates a foundation for dynamic analysis that can further reveal subtle defects or vulnerabilities
12	Codereviews	▪ Peer code reviews	▪ Helps improve integrity of the code change prior to check-in
13	Secrets management	• Protects secrets needed to access your applications, services, and ITresources • Enables users to easily rotate, manage, and retrieve database credentials, API keys, and other secrets throughout theirlifecycle	

Market Leading Tools*

There are many tools to enable version control and many of them are used across Deloitte projects. Listed below a have a strong market presence and used predominantly within Deloitte technology practice:

Market Leading Tools:

- ***Atlassian Bitbucket***
- ***Collabnet Subversion***
- ***IBM Rational ClearCase***
- ***Microsoft Team Foundation Server (TFS)***
- ***Microfocus PVCS***
- ***Git***
- ***Perforce Helix TeamHub***
- ***GitHub Enterprise***

Listed below are some of the popular freeware version control tools in the market:

- ***Collabnet Subversion***
- ***Git***
- ***CVS***
- ***Mercurial***

*** Note :- For more details on tool features and integrations please refer Source Code Management Point of View**

Listed here is the common functionality among the SCM tools and their comparison, which would help determine the major differences among them.

		Atlassian Bitbucket	Collabnet Subversion	IBM Rational ClearCase	Microsoft TFS	Microfocus PVCS	Git	Helix Team Hub	GitHub Enterprise
Repositories	Supports central repository	√	√	√	√	√	√	√	√
	Supports distributed repository	√	X	X	X	X	√	√	√
	Supports multi-tenancy and fine-grain access control	√	√	√	√	√	√	√	√
	Supports notifications	√	√	√	√	X	√	√	√
	Provides good support for security	√	√	√	√	√	√	√	√
Versioning	Version control of files and workspace management	√	√	√	√	√	√	√	√
	Supports comparison of file versions and highlights differences	√	√	√	√	√	√	√	√
	Maintains audit trail of changes	√	√	√	√	√	√	√	√
	Supports versioning of directories and directory properties, just like files	X	√	X	X	X	X	X	X
Branching and Merging	Supports branching to manage multiple releases	√	√	√	√	√	√	√	√
	Supports merging or consolidation of code bases	√	√	√	√	√	√	√	√
	Supports workflows for process automation and merging	√	√	√	√	√	√	√	√
	Peer code review	√	√	√	√	√	√	√	√
	Supports forking to create a copy of the repository	√	X	X	√*	X	√	√	√
Integration	Can be integrated into all major IDE's	√	√	√	√	√	√	√	√
	Can be integrated with all major build management systems	√	√	√	√	√	√	√	√
	Provides out of the box mechanism for authentication and authorization, integrates with LDAP, AD and has single sign of capability	√	√	√	√	√	√	√	√

Legend
√ Supported
X Not supported

* TFS when used with Git Virtual file system supports Git forking features

Continuous Integration

Continuous Integration (CI) is a process wherein developers regularly integrate their code changes into a shared repository. Automated builds and tests are run after each check-in to validate the software quality quickly and report any issues early in the software development life cycle.

How it works ?

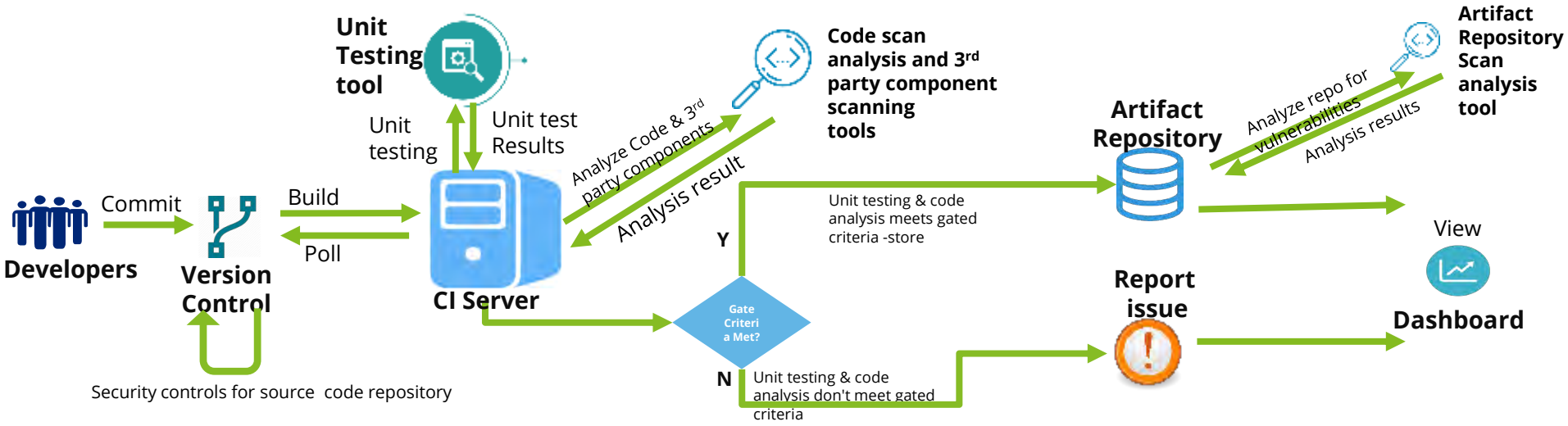
It is a series of iterated processes specifically for application development phase in which:

- Code changes are regularly merged into a **repository** for validating the compatibility
- **Builds** are automatically created and **tests** are performed
- This allows for the early detection of bugs and errors. When corrections happen earlier in the SDLC, costs associated with fixing these issues decay exponentially.

Fundamental Elements

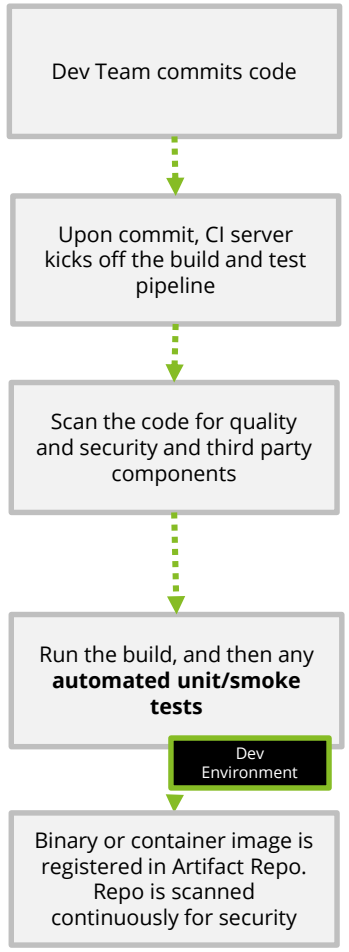
- Regular commits and resolutions to conflicts
- Code Repositories with proper Branching Strategy
- Security controls provided for source code repository
- CI Server to detect code changes which trigger build and test execution
- Repository to store artifacts
- Ability to monitor CI pipeline and builds
- Artifact repository scanning for security vulnerabilities in binary files
- Continuous third party component scanning

A Standard Continuous Integration System



Continuous Integration in Action

A Standard CI Workflow



Core CI Capabilities

Automated Build is the process of automating the creation of a software build and the associated processes including compiling source code into a deployable unit like binary or container image, packaging the deployable unit, and running automated unit tests.

Automated Unit Tests tests the smallest unit of code (method / class, and so on) that can be tested in isolation from other units. Running the unit test frequently helps in maintaining code health and finding errors before code is deployed to production.

Code Quality Scanning tools analyze code structure and ensure it adheres to industry standards of quality and security. The outputs from these tests can be channeled backed into CI servers and used to control conditional deployments. **Third part component scanning** helps automatically identify open source components in the product, gives real-time alerts on open source vulnerabilities & helps in maintaining licenses for third-party components

Automated Functional Tests tests the entire application for existing or new functionality in a fully deployed version of the application in an environment, ensuring quality of code before being deployed to production.

Artifact Repository optimize the download and storage of artifacts used and produced in software development process. It centralizes the management and **security** of all the artifacts generated and used during the CI process. Scan Git repositories for secrets, keys & credentials. The entire commit history and all branches are scanned, in order to catch any secrets accidentally committed. This is done through eithers canning for keywords or calculating Shannon entropy of strings.

Automated Unit Testing

Automated unit testing is one of the first integrated stages in a DevOps pipeline, with build failure conditions set for most pipeline patterns

What are Unit Tests?

An **Unit Test** is a test of the smallest unit of code (method / class, and so on) that can be tested in isolation from other units. Running the unit test frequently helps in maintaining code health, ensuring code coverage, and finding errors / faults before code is deployed to production.

An Unit Test:

- Should **not** require your **application running** or **access the database**
- Leverages **xUnit style framework** to run against your binaries
- **Consistent** (same results) and **atomic** (T/F)
- **Single Responsibility** – test one behavior only; methods with multiple behaviors = multiple tests
- **No conditional logic or loops**
- Gets organized in suites and **executed in parallel by a CI server** as applications grow
- **When fails, fails the entire build**

```
 9      @Test
10      public void testCalculator() {
11          Calculator calculator = new Calculator();
12          final int add = calculator.add(1, 2);
13          assertThat(add, Is.is(3));
14
15          final int subtract = calculator.subtract(2, 1);
16          assertThat(subtract, Is.is(1));
17          final int another = calculator.add(subtract, 1);
18          assertThat(another, Is.is(2));
19      }
20
21  }
22
```

Automated **Unit Testing** is the foundation of **Test-driven Development (TDD)*** where developers, before writing any functional code, create a test that is an executable specification of the expected behavior of code intended to be written. It not only builds quality, but also influences application design and creates documentation. **It's a balance between discipline and pragmatism**.

*** Note :- For more details on TDD and BDD please refer Continuous Testing Development Guidelines**

What is Code Quality?*

Code Quality is an approximation of long-term usefulness and maintainability of code. Code Quality has three main sub-capabilities: Code Analysis, Security Analysis, and Code Review.

These sub-capabilities taken together determine if the system has been built correctly based on the application's compliance to best practices and its current quality and security standards.

Code Quality	=	Code Analysis	+	Security Analysis	+	Code Review
Addresses the question of "Has the system been built right?"		Identify issues related to programming practices		Identify security vulnerabilities of the solution		Identify compliance to standards defined & best practices

*** Note :- For more details on Code Quality please refer Code Quality Tools POV**

Static Code Analysis for Code Quality

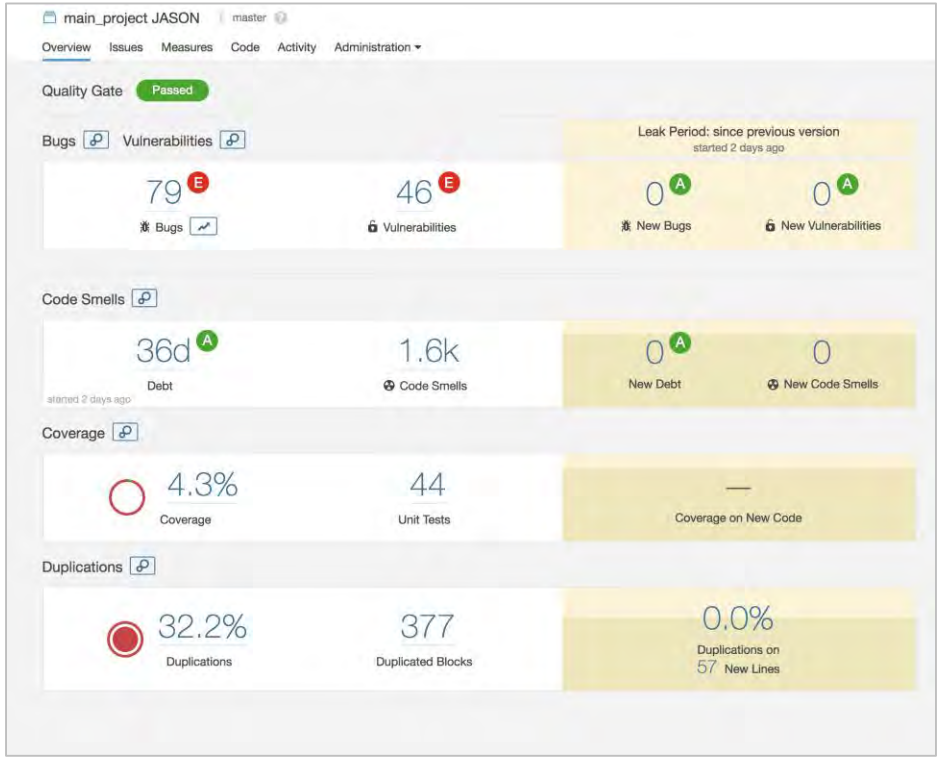
Static code analysis is a collection of algorithms and techniques used to analyze source code in order to automatically find potential errors, security vulnerabilities or poor coding practices.

Standard output of static code analysis :

- Bugs
- Code smells
- Code coverage
- Test coverage
- Duplicate code blocks due to bad design

Static code analysis enables:

- Continuous inspection of overall health
- Centralized code quality management and standards
- Conditional deployments via test result integration with CI server



Artifact Repository

Artifact Repositories like Nexus or Artifactory are designed to optimize the download and storage of artifacts like binaries, packages, container images, libraries, used or produced in a software development process.

Why use a tool for Artifact Management?

- Single source of truth for all components
- Improved CI/CD outcomes with a single repository to manage all assets related to development and delivery
- Improving performance, **security** and stability for builds and other component users
- Effective policy management of components to ensure secured use of 3rd party components and libraries
- Ability to **sign container images** with newer repo managers, like docker trusted registry

Leading Practices

- Separate repositories for separate technology stacks (e.g., Java, .NET, Node, etc.)
- Virtual repository for each of your team / builds
- Use Include / Exclude content features to effectively across internal and external content
- Single maven/npm repos for all projects

Wrapping security around Continuous Integration

Align security tasks directly with development, build, and testing, providing self-service capabilities and immediate feedback where possible through tooling.

Build

In-line automated security testing prior to code commit

Development

- Risk evaluation of change
- Focused SAST/DAST (OWASP 10 / SANS 25)
- Security unit tests (positive & negative)
- Dependency/license/open source

DevOps Toolchain

- Pre-commit hooks
- IDE plug-ins
- Integrated build orchestration
- Quality Gate/Break the build criteria

Infrastructure

- Container / VM security
- Infrastructure config analysis (linting)
- Use VPC to segregate environments

Quick Nuggets

- Build ruleset of tests across key areas such as OWASP Top 10 and SANS Top 25
- Incorporate Static and Dynamic Application Security testing (SAST and DAST)
- Establish Quality Gate i.e. pass/fail criteria and transparent results

Continuous Integration: Leading Practices (1 of 3)

DevOps Processes and capabilities across the value stream

Core Capabilities

#	Leading Practice	Key Considerations	Key Benefits
1	Multiple Small-Sized Code Commits (at least once a day)	▪ Avoid multiple branches. The effort required to successfully merge branches increases exponentially as the number of branches increases. As the number of developers increases, the probability that any given change will break the build also increases. Therefore, keep number of branches to a minimum. ▪ To avoid merge complexities and minimize code refactoring, make sure developers make multiple code commits (eventually in the range of hundreds per day).	▪ Reduces the probability of breaking builds/failing tests ▪ Ensures faster feedback
2	Check-ins Trigger Automated Builds with Prioritized CI Test Cases	▪ CI design should focus on trunk-based development. ▪ CI builds should be configured to trigger on every code check-in, and they should include execution of the automated tests to ensure that every code check-in does not impact the trunk's releasable state.	▪ Team productivity optimization increases the flow of work in the value stream ▪ Many developers can independently develop, test, and deliver value
3	Build Status is Visible to All	▪ Because developers develop on a single trunk, it is critical to maintain the transparency of the build status. ▪ When a build fails, the deployment pipeline should block all subsequent commits, thereby forcing all developers to stop what they were doing and swarm together to address the issue that is affecting the build. This ensures transparency across team in terms of the build status.	▪ Team productivity optimization increases the flow of work in the value stream
4	System Architecture*	▪ Services as a deployable unit should be mutually exclusive; independently deployable, testable, and scalable; and isolated from failures of other upstream or downstream application/services . ▪ Application/services must be stateless and not coupled to the underlying OS or infrastructure. APIs should be designed in collaboration with the developers of downstream teams. ▪ To ensure data isolation and to adhere to the loosely coupled architecture, application/services do not necessarily share database technologies or a logical database and schemas. Applications should be built with portability in mind (e.g., portability to cloud). ▪ Teams are free to use any languages, architectures, dependencies, and databases needed without external approval as long as it does not violate overall IT principles or guidelines.	▪ Ability to deliver a functional value by serving specific business domains (e.g., orders, inventory, etc.)

*** Definitions are provided in the Additional Resources.**

Continuous Integration: Leading Practices (2 of 3)

DevOps Processes and capabilities across the value stream

Advanced Capabilities

#	Leading Practice	Key Considerations	Key Benefits
5	Builds and Prioritized CI Test Cases Run in Minutes	▪ As part of Continuous Integration builds, execute these automated tests: ▪ Unit Tests: Unit tests should execute within minutes and triggered on every check-in ▪ Automated Acceptance Tests: Executed on every commit, every two to four hours ▪ Full Regression Tests: Execute weekly/Bi-weekly	▪ Ensures faster feedback ▪ Proactively detect issues before they are discovered in formal testing phase by QA or business users
6	Automatic Notification of Build and CI Test Case Failures	▪ If a code change causes CI build or any automated tests to fail, the developer should be automatically notified so corrections can be made without affecting others in the value stream.	▪ Delivers feedback faster so corrective steps can be taken as early in the process as possible
7	Reject Commits that take Trunk out of Deployable State	▪ Configure the deployment pipeline to reject any commits (i.e., code or environment changes) that remove the trunk from a deployable state. A trunk should maintain a code that is always buildable and deployable in state. Any changes that divert the trunk to a non-deployable state should not be allowed to enter the trunk. ▪ Employ gated commits in the deployment pipeline to ensure that submitted changes can be built successfully and pass all automated tests before being merged into trunk.	▪ Facilitates trunk-based development by always maintaining the trunk in deployable state
8*	BVT Automation with Static Analysis Code Coverage Results	▪ As part of the build verification test (BVT), execute a set of pre-defined test cases on each new build to ensure the stability of the build for further testing.	▪ Identify build issues quickly ▪ Automates review for code quality, security, and compliance ▪ Ensures appropriate code coverage for automated test cases executed as part of the build

*** Definitions are provided in the Additional Resources.**

Continuous Integration: Leading Practices (3 of 3)

DevOps Processes and capabilities across the value stream

Security Capabilities

#	Leading Practice	Key Considerations	Key Benefits
9	Container security	▪ Provides visibility and control over containerized environments, with tightruntime security controls, image assurance, and intrusion preventioncapabilities	
10	Continuous third-party component scanning	• Automatically identifies open source components in theproduct • Gives real-time alerts on open sourcevulnerabilities	▪ Helps in maintaining licenses for third-party components
11	Config management, Infrastructure provisioningand scanning	• Static code scan which identifies vulnerabilities and misconfigurations in the scripts/cookbooks used to provision infrastructure dynamically	

*** Definitions are provided in the Additional Resources.**

Market Leading Tools*

These are leaders in enabling Continuous Integration.

Market leading tools evaluated:

- ***GitLab CI***
- ***Cloudbees Jenkins Enterprise***
- ***Microsoft Visual Studio Team Services***
- ***CircleCI***

In addition there are freeware CI tools listed below that may meet the needs of some projects:

- ***Jenkins***
- ***Travis CI***

Note :-

1. **For more details on tool features and integrations please refer Continuous Integration Point of View**
2. **Tools from Delivery Excellence available for client projects ALM, JRebel, Fortify, & SonarQube**

The following table provides a list of categories and common features used to evaluate each of the leading market tools

		GitLab	Jenkins	Microsoft	Circle CI
Build Management	Supports multiple build engines	✓	✓	✓	✓
	Scheduling builds	✓	✓	✓	✓
	Post-build triggers	✓	✓	✓	✓
	Poll SCM	✓	✓	✓	✓
Artifact Management	Maintain versions of build/code artifacts	✓	✓	✓	✓
	Historical information and archiving	✓	✓	✓	✓
	Ability to rollback builds	✓	✓	✓	✘
Reporting	Build status reports	✓	✓	✓	✓
	Notifications	✓	✓	✓	✓
	Enhanced Reporting	✓	✓	✓	✓
Integration	Integration with SCM, Code Analysis, Defect Management and Deployment solutions	✓	✓	✓	✓
	Integration with major IDEs	✓	✓	✓	✘

Legend
✓ supported
✘ not supported

Continuous Testing

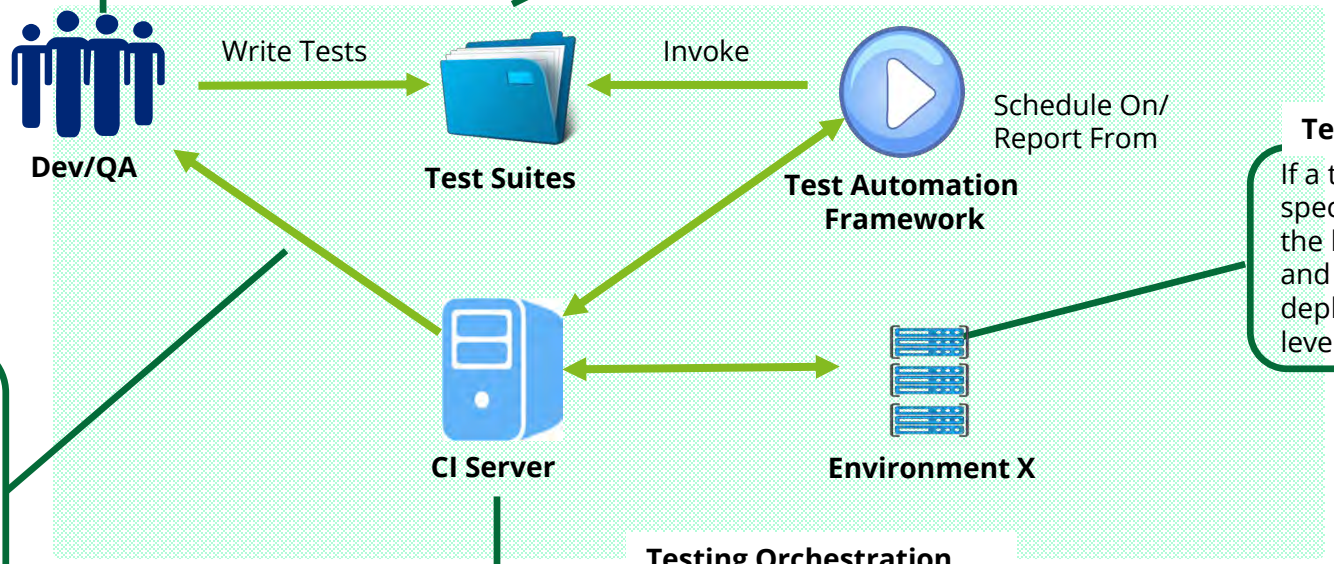
Continuous testing (CT) is the process of executing automated tests in a continuous way throughout the development life cycle to provide immediate feedback on application quality.

Test Case Creation

Collaborating with developers, QA creates automated test cases. These tests are stored and invoked using the same tools and pipeline which developers use to build and deploy code.

Test Suite Organization

Test are segmented into different suites, which are executed in different environments, with varying degrees of complexity and success thresholds.



Test Certificationn

If a test meets the specified passing criteria, the build is then labeled and eligible for deployment into higher level environments.

Learning

The insight gained from the automated analysis of the data generated by the execution of the test suite is delivered back to developers and testers. Bugs and defects can then be prioritized for future sprints.

Testing Orchestration

The tests which are invoked by the Continuous Integration server generates vast amounts of data. Using the correct tools and processes this data can be analyzed to generate insights which go beyond a simple pass or fail status. Proper analysis can allow for root cause analysis into failures and provide recommendations for possible solutions.

Progressive Testing Pipeline Patterns

Progressive testing pipeline segments tests into suites which are automatically run at different frequencies and in different environments. The tests in each suite gets progressively more complex as a build transverses into higher level environments.

	Unit	Sanity	Targeted/ Risk Based	Full	Non functional
Type of test	Gunit, Junit	Application sign of life and basic functionality tests	Selected Key functionality tests	Full Regression	Security and performance tests
Objective	Testing smallest possible unit(Code) in isolation	Verify successful deployment and no connectivity issues	Verify key functions are working as designed	Verify key functions are working as designed	Verify security vulnerabilities and performance
Threshold	80-90%	95%	90%	80%	No Critical or Highs
Execution Time	~ 1 minute	10 minutes	1 hour	4 to 12 hours	4 to12 Hours
Execution Frequency	Every Build	4 times of day	Daily	Weekly/Bi- weekly	*SAST – Daily DAST/IAST/Perf-Weekly/Bi-Weekly

Continuous Integration

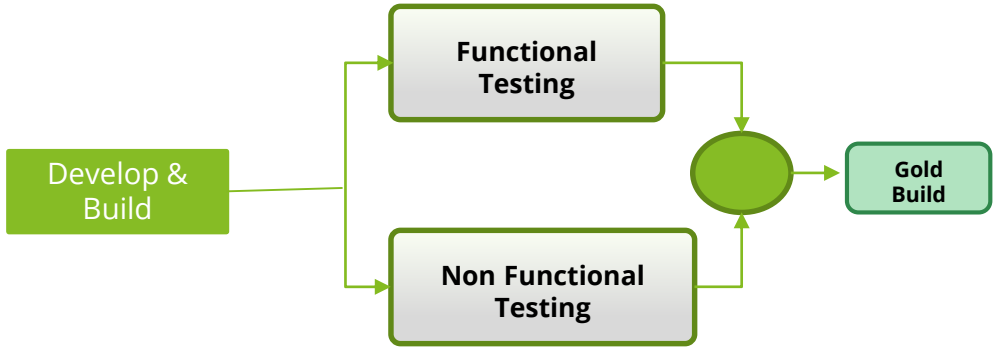
✓ Latest Build → ✓ Successful Build → ✓ Stable Build → ✓ Solid Build → ✓ Certified Build → ✓ Gold Build

*SAST – Static Application Security Test, DAST – Dynamic Application Security Test, IAST – Interactive Application Security Test

Advanced Testing Pipelines

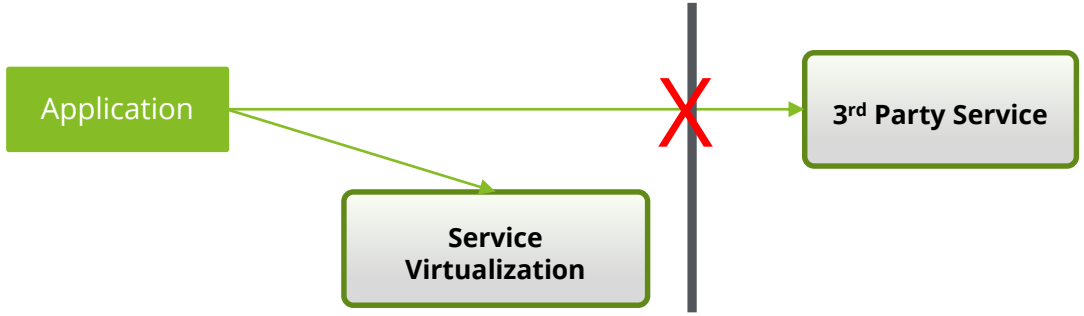
Parallel Test Pipelines

- Multiple testing pipelines which are initiated concurrently and test different facets of an applications, for example functional and non-function.
- Conditions can be implemented which only allow builds to be promoted which successful pass in both testing pipeline.



Service Virtualization

- Virtualizing key services allows the test automation team to create scripts before the development team has completed the feature.
- Virtualization of third party services can reduces costs and risks of environment unavailability.



Continuous Testing: Leading Practices (1 of 3)

DevOps Processes and capabilities across the value stream

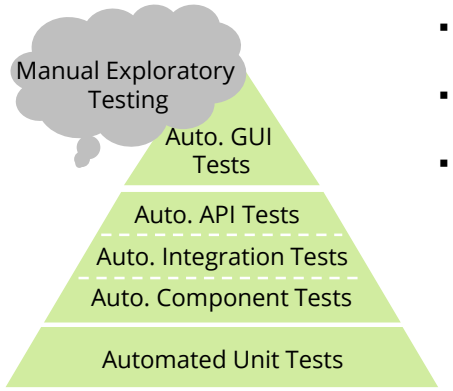
Core Capabilities

#	Leading Practice	Key Considerations	Key Benefits
1	Production-Like Test Environment	▪ It is important to verify that the application runs as expected in a production-like environment before deploying it to production; this requirement should be added to the definition of "done" for the development.	▪ Improves the reliability of testing process and confidence in production releases ▪ Integration of code and environment happens as part of daily work instead at the end of the release
2	Comprehensive Automated Tests Validate Releasable State	▪ To avoid frequent breaking of builds and failure of automated test cases, perform integration and testing steps continuously, not periodically. ▪ Prevent this scenario by having fast, automated tests that run within the build and test environments whenever a new change is introduced into version control.	▪ Allows immediate detection and resolution of any problems ▪ Ensures that batches remain small ▪ The trunk remains in a deployable state
3	Automated Acceptance and Regression Tests	▪ The objective of the acceptance test is to prove that the application does what the customer meant it to and not that it works the way programmers think it should. ▪ After a build passes unit tests, the deployment pipeline should execute automated acceptance tests. Once those test are passed, build can be made available for any manual testing. ▪ To maximize the ability to find the defects during acceptance testing, use virtual or simulated versions of remote services.	▪ Ensures that application features meet the acceptance criteria articulated in respective stories ▪ Fulfills some of the criteria for certification from the business users for subsequent deployment in production
4	Automated Integration Tests	▪ Integration tests verify that the application correctly interacts with other production applications and services, as opposed to calling stubbed-out interfaces. ▪ Integration tests are performed on builds that have passed unit and acceptance tests. ▪ Minimize the number of integration tests because they are often brittle. Many of the defects should be found during unit and acceptance testing.	▪ Verifies functioning of external integrations and dependencies against the mocks/stubs tested in unit/component testing level

Continuous Testing: Leading Practices (2 of 3)

DevOps Processes and capabilities across the value stream

Advanced Capabilities

#	Leading Practice	Key Considerations	Key Benefits
5	Ideal Testing Pyramid (Unit > Integration > GUI > Manual Tests)	▪ Martin Fowler described the ideal testing pyramid as follows:  ▪ Ideally, most issues should be discovered in the unit tests. ▪ In contrast, many testing programs make most investments in manual and integration testing. ▪ If unit or acceptance tests are too difficult and expensive to write/maintain, the architecture is monolithic and tightly coupled. It probably needs to be refactored to create more loosely coupled modules that can be tested independently without needing an integrated environment.	▪ Faster running automated tests help find errors as early as possible ▪ Fixes are cheaper and faster when errors are detected early in the cycle
6	Tests Execute in Parallel and Complete Quickly	▪ To run the test suite frequently, it needs to complete quickly. Therefore, tests should be executable in parallel, potentially across multiple servers. ▪ Different types of tests should be run in parallel, e.g., when build passes acceptance tests, security and performance tests can be executed in parallel. ▪ Wait for automated tests to successfully complete before allowing manual exploratory testing. This enables testers to test the right builds and removes their dependency on developers to flag a particular build to test.	▪ Enables continuous testing as frequently and practically as possible ▪ Facilitates testing as early in the process as possible
7*	Integrated Performance and Other Non-Functional Testing	▪ Use integrated performance and other non-functional testing in the deployment pipeline to improve capacity planning and to detect conditions that can result a spike in resource utilization.	▪ Validates the consistency and accuracy of environment configurations to meet the required non-functional requirements

*** Definitions are provided in the Additional Resources.**

Continuous Testing: Leading Practices (3 of 3)

DevOps Processes and capabilities across the value stream

DevOps Processes
- Version Control
- Continuous Integration
- Continuous Testing
  - Basics
  - Leading Practices
  - Tools
- Continuous Deployment
- Continuous Monitoring

Security Capabilities

#	Leading Practice	Key Considerations	Key Benefits
9	Static Application Security Testing ("SAST")	• Analyzes application source code, byte code, and binaries to detectvulnerabilities • Some SAST tools can be integrated directly into the developmentenvironment	▪ Technology analyzes application source, byte or binary code for security vulnerabilities like cross site scripting, SQL Injection, Password Management, etc.
11	Interactive Application Security Testing ("IAST") & Dynamic Application Security Testing ("DAST")	• IAST analyzing application behavior during test / QA ("quality assurance") by using runtime application self-protection ("RASP") agent, which instruments software, and dynamic application security testing ("DAST") to introduceattacks • IAST tools can integrate directly in the continuous integration / continuous delivery ("CI/CD") pipeline and return results as soon as new code is recompiled and the application is tested, enabling developers to detect vulnerabilities earlier in the development cycle	▪ IAST is typically implemented as an agent within the test runtime environment that observes possible attacks and identify a sequence of instructions that could lead to an exploitation ▪ DAST analyzes applications in their running state and simulates attacks like penetration testing, access elevation, etc. to determine security vulnerabilities
12	Penetration testing	• Periodic manual penetration testing to test implemented securitycontrols	

*** Definitions are provided in the Additional Resources.**

Market leading tools*

These tools are leaders in enabling automated testing.

Market Leading Tools :

- ***Micro Focus Unified Functional Testing (UFT)***
- ***Micro Focus LeanFT***
- ***Micro Focus Silk Test***
- ***Tricentis Tosca***
- ***Microsoft Test Manager***
- ***IBM Rational Test Workbench (RTW)***
- ***SmartBear TestComplete***
- ***CA Technologies suite***
- ***TestPlant eggPlant Functional***
- ***Blue Prism***

Additionally, several free/freemium tools are also available in the market that offer a less expensive alternative, which may be suitable for use depending on the specific project or client need.

- ***Selenium***
- ***JUnit***

Note :-

1. **For more details on tool features and integrations please refer Testing capability based POVs**
2. **CT Tools from Delivery Excellence available for client projects -Performance Center, Unified Functional Testing and UFT Pro**

The following table provides a list of categories and common features used to evaluate some of the leading market tools

		Micro Focus UFT	Micro Focus LeanFT	Micro Focus Silk Test	Tricentis Tosca
Usability	Ease of test script creation; intuitive UI, wizards and dialogs available	✓	✓	✓	✓
	Configuration of test scenarios and test schedule	via ALM	✗	✓	✓
	Framework: keyword-driven, data-driven, hybrid	✓	✓	✓	✓
Execution	Test case management facility	via ALM	✗	✗	✓
	Debugging support	✓	✓	✓	✓
	Analyzing test results	✓	✓	✓	✓
Reporting	Notifications	✓	✓	✓	✓
	Documenting and publishing test results	✓	✓	✓	✓
	Customizable reporting	✓	✓	✓	✓
Integration	Integration with other tools like IDEs, Build Tools, Micro Focus ALM & AgM	✓	✓	✓	✓
	Version control adaptability	✗	✗	✓	✓
	Cross-browser support	✓	✓	✓	✓
	Cross-platform support	✓	✓	✓	✓

- **Products that require add-ins or integration with other tools to perform described functionality are listed as not supported.**

Legend
✓ = supported
✗ = not supported

The following table provides a list of categories and common features used to evaluate each of the leading market tools***(continued)***

		TestPlant eggPlant	Blue Prism	Selenium	JUnit
Usability	Ease of test script creation; intuitive UI, wizards and dialogs available	✓	✓	✗	✗
	Configuration of test scenarios and test schedule	✓	✓	✓	✗
	Framework: keyword-driven, data-driven, hybrid	✓	✗	✓	✓
Execution	Test case management facility	✓	✗	✓	✗
	Debugging support	✓	✓	✓	✓
	Analyzing test results	✓	✗	X	✗
Reporting	Notifications	✓	✗	✓	✗
	Documenting and publishing test results	✓	✗	✗	✗
	Customizable reporting	✓	✗	✗	✓
Integration	Integration with other tools like IDEs, Build Tools, Micro Focus ALM & AgM	✓	✗	✓	✓
	Version control adaptability	✗	✗	✗	✗
	Cross-browser support	✓	✗	✓	✓
	Cross-platform support	✓	✓	✗	✓

- **Products that require add-ins or integration with other tools to perform described functionality are listed as not supported.**

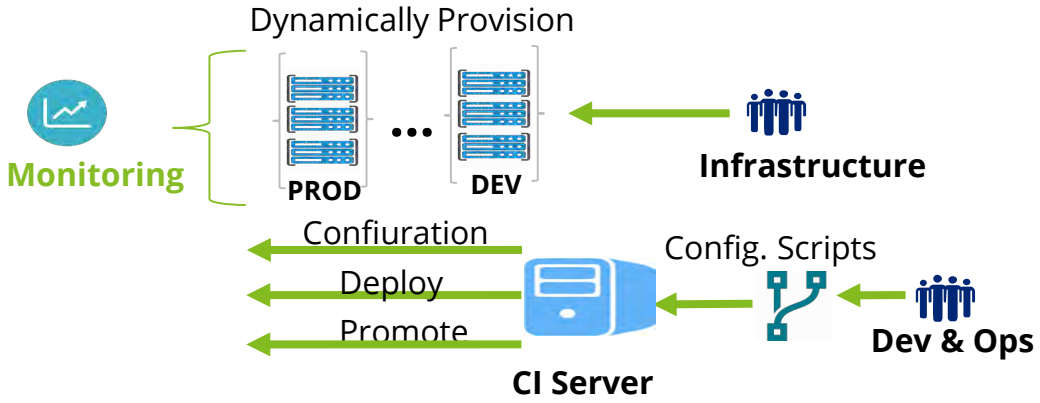
Legend
✓ = supported
✗ = not supported

Continuous Deployment

Continuous Delivery means that you are ready and able to deploy any version to any supported platform at any time, whereas **Continuous Deployment** means that you are engaging in actual deployment

Fundamental Elements:

- Infrastructure is dynamically provisioned
- Configuration automated and scripts version controlled
- Clear visibility into the health of all environments



Advanced Deployment Patterns

- Blue / Green Deployments
- Canary Releases
- Rolling Updates
- Dark Launches and Feature Toggles

Principles of Low-risk Releases

- Low-risk Releases are Incremental
- Decouple Deployment and Release
- Focus on Reducing Batch Size
- Optimize for Resilience

Release Automation and Orchestration

Continuous Delivery is not just "a few more jobs in my CI server". It adds the equally important process integration across Change, Risk & Compliance, Release, and Project Management.

How does a Release get orchestrated?

1. Create a model-based release **template that manages all of the tools in your build, test and deploy process**, starting with polling SCM to pick up developer changes, then build, deploy and test your application in the lower environments

2. Create an model-based release template that **manages all of the tasks, both automated and manual**, for delivering your application, including application and release dependencies.

3. When the team says that a release is ready, **kick off the Production release template**, either in an automated way (updating a ticket) or a manual one (e-mail to PM)

Pipeline can be managed by people who are not automation experts

provides an easy to use interface for Dev, QA, DBAs, release managers and other business users, not just automation gurus..

Meet compliance and audit requirements

Automatically provides security and audit trail for all changes to pipeline definitions and running releases to ensure you meet compliance and auditability requirements for the entire process.

Integrate with all tools

Integrate with service, change and incident management systems, document management systems, or project and program management products, not just build and test tools.

Visualize your Pipeline

Get the powerful analytics you need to improve your processes using a simple reporting API: no proprietary language, no scripts, no need for Groovy DSLs.

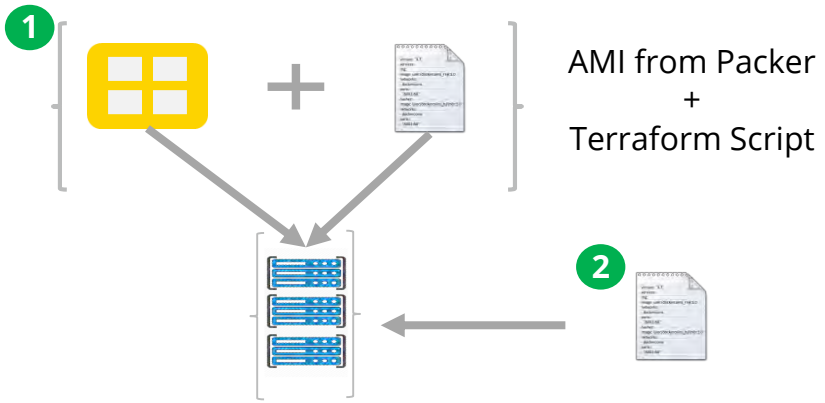
Automated Infra Provisioning and Configuration Management

Terraform and Ansible broadly fall in the "Infrastructure-as-code" category where they are not mutually exclusive. Both set of tools have some capabilities of the other tool. Terraform can do some infra provisioning and Ansible can do some config management.

Automated Infra Provisioning

- **Provision servers** with declarative code, leaving most of server configuring job to other tools
- **Write**, **plan** and **create** reproducible infrastructure
- Uses immutable **Machine Images** as an input, "baked" by tools like Packer
- Immutable container image -> Running container

  Immutable machine image -> Running server



Automated Config Management

- Install and manage software on existing or automatically provisioned servers
- Another form of infrastructure as code, managed in the form Chef cookbooks or Ansible playbooks
- **Example**: Installing and configuring OpenShift cluster on a set of automatically provisioned AWS VM nodes

Packer Terraform ANSIBLE

1 A server instance is automatically provisioned by using a terraform script with an immutable machine image from Packer as input

2 Ansible playbooks are run to on the provisioned server to apply additional software configuration resulting in a set of environments with minimal **configuration drifts**

Wrapping security around Continuous Deployment

Protect the integrity and agility of the deployment pipeline while also performing risk rationalized security testing steps.

Release/Deploy

Risk rationalized checks before, during, and after code is deployed

Development

- Risk-based release strategy and associated security tests
  - Additional SAST/DAST tests
  - Penetration tests
  - Security smoke tests
  - Security acceptances tests
- Production controls hardening

DevOps Toolchain

- Integration of security tools into release gate

Infrastructure

- Risk-based release strategy and associated testing
- Production controls hardening
- Security smoke tests
- Manage changes to container / VM baseline

Quick Nuggets

- Build hardened baselines and safe guard gold copies and templates
- Vulnerability assessment of infrastructure
- Manual code review, architecture review, and penetration testing of high-risk applications or functionality

Continuous Deployment: Leading Practices (1 of 3)

DevOps Processes and capabilities across the value stream

Core Capabilities

#	Leading Practice	Key Considerations	Key Benefits
1	Automated Packaging for Deployment to any Environment	▪ Same deployment mechanism must be used for every environment (e.g., development, QA, pre-production, production). ▪ Any build should be deployable to any of the environments by ensuring that the source code has no environment-specific configurations. ▪ All environment-specific configurations that the application needs are stored outside the source code in version control (i.e., within the environment configuration scripts that are used to build the environments).	▪ Improves the predictability of production deployment ▪ Improves the repeatability of all pre-production and production deployments
2	Self-Service, Automated Deployment and Configuration	▪ To achieve the DevOps outcomes, shift reliance to control mechanisms (like automated testing, automated deployment and peer review of code changes) to drive transparency, responsibility, and accountability between developers and operations. ▪ Either developers or operations should perform the code promotion process with minimum manual activity: (1) **Build**: Packages should be deployable to any environment; (2) **Test**: Anyone should be able to run any or all automated tests; and (3) **Deploy**: Anyone with access should be able to deploy packages to any environment executed by running scripts checked in to version control. ▪ Self-service tools are available for developers and testers to procure production-like environments on demand based on standard templates that are defined by organization. They should be available to configure new build procedures and pipelines on demand to create new development branches. ▪ Deployment to Test and Staging environments is automated and self-service.	▪ Accelerates flow from development to deployment ▪ Enables successful deployments regardless of who is performing it ▪ Enables deployments without the user having elevated access on production servers or access to sensitive production data
3	Automated Smoke Tests and Result Notifications	▪ During the deployment process, testing should be conducted to ensure connectivity to any supporting systems (e.g., databases, message buses, external services) and to run a single test transaction through the system to ensure that systems are performing as designed. ▪ If any of these tests fail, deployment should be failed and rollback should be initiated.	▪ Proactive detection and resolution of catastrophic issues before end users are affected

Advanced Capabilities

#	Leading Practice	Key Considerations	Key Benefits
4	Pre-Configured Virtual Machine Images or Containers	▪ Application containerization is an OS-level virtualization method and should be used to deploy and run distributed applications without launching an entire virtual machine for each application. ▪ Multiple isolated applications or services should be able run on a single host and access the same OS kernel.	▪ Improves memory, CPU, and storage efficiency when compared to traditional virtualization ▪ Without the overhead for VMs, many more application containers can be supported on the same infrastructure
5*	Automated Rollback in case of Deployment Failures	▪ Production monitoring systems that can detect failures or abnormal behaviors can be configured to provide it as a feedback to deployment pipeline, which can immediately deploy previous known "good" build (i.e., rollback of new code).	▪ Protects against the defects that are hard to find using automated tests ▪ Reduces the time required to detect and respond to the degraded performance created by the new change
6	Deployment Strategies (e.g., Feature Toggle, Canary)	▪ Use Feature Toggle to hide, enable, or disable features during runtime so the feature can be tested/deployed before it is completed. ▪ Use Canary Release to reduce the risk of introducing a new software version in production by slowly rolling out the change to a small subset of users before rolling it out to the entire infrastructure and making it available to everybody.	▪ Work on a trunk without creating branches ▪ Provides a rapid rollback option if issues arise ▪ Minimizes risks if issues arise when deploying the new feature

7	Zero Downtime Deployments	▪ Decoupling deployments from releases dramatically changes weekend releases. Deployments do not need to be performed in the middle of the night or over the weekend to lower the customer impact.
		▪ Deployments can be performed during typical business hours by leveraging certain patterns:
		  ▪ For Blue-Green Deployment approach you should have two production environments that are as identical as possible. At any time, one of them, say blue, is live. As you prepare a new release, you do your final stage of testing in the green environment. Once the software is working in the green environment, you switch the router so that all incoming requests go to the green environment.
		  ▪ Database Changes: 1) Create two databases: blue and green. Set one (e.g., blue) to read-only mode. Take backup and restore it into green. Deploy code to green, and then direct traffic to green. If something goes wrong, redirect to blue (i.e., rollback); 2) decouple database changes from application changes so the database changes can be released first (in a way that it does not impact the users) and then application changes following the B/D model.
			▪ Enables deployments during normal business hours; simple changeovers (e.g., router setting, LB redirection) can be conducted during off-peak hours
			▪ Improves work conditions for the team performing the deployment
			▪ Fosters a continuous deployment culture by reducing batch size

*** Definitions are provided in the Additional Resources.**

Advanced Capabilities

#	Leading Practice	Key Considerations	Key Benefits
8	Release Planning/ Prioritization, Metrics, and Dashboards	▪ Regularly collect customer feedback about the design and quality of features. This should be used as input for the prioritization and design of features. ▪ Developers and Product Owners should track and watch how consumers interact with their application/services. ▪ Prioritize non-functional requirements using usage patterns observed, information from transaction monitoring, and insights from analytics tools. ▪ As part of Sprint planning and capacity allocation, account for enough time within sprints for developers to focus on non-functional requirements and technical debt (such as refactoring, automation, and architecture). ▪ Break down features into smaller viable units of work that can be deployed in a sprint or shorter period of time. ▪ Limit work-in-progress items at every stage of the delivery cycle (i.e., development, testing, and deployment). ▪ To manage value flow better, allocate a portion of capacity to unplanned work and firefighting work based on delivery history/retrospective. ▪ Dashboards are available for product and release quality, technical debt information (product release level), code coverage metrics (application release level), and security metrics (product release level).	▪ Helps improve ease of user interaction and solution testability
9	Approval to production	▪ Business owner approval prior to releasing changes to production to increase visibility and mitigate business disruption ▪ Expands focus of development beyond code to the operating environment, including infrastructure, network, and application dependencies ▪ Increases accountability of teams responsible for changes in production to improve quality and increase reliability and maintainability of code	

Market Leading Tools*

There are various software deployment tools available in the market and listed below are the tools considered for POV, shortlisted based on market leaders identified by Forrester report (Forrester Wave: Application Release Automation Q3'17) and usage within Deloitte:

Market Leading Deployment Tools evaluated:

- ***XebiaLabs XL Deploy***
- ***IBM UrbanCode Deploy***
- ***ElectricFlow Deploy***

Market Leading Release Automation Tool evaluated

- ***CA Release Automation***

Additionally, several free/freemium tools are also available in the market that offer a expensive alternative, which may be suitable for use depending on the specific project or client need. Note – clients should carefully consider the use of free tools based on their security policy, data residency and data privacy policies. Some of the configuration management & environment related tools are:

- ***Chef***
- ***Docker***

*** Note :- For more details on tool features and integrations please refer Continuous Deployment Point of View**

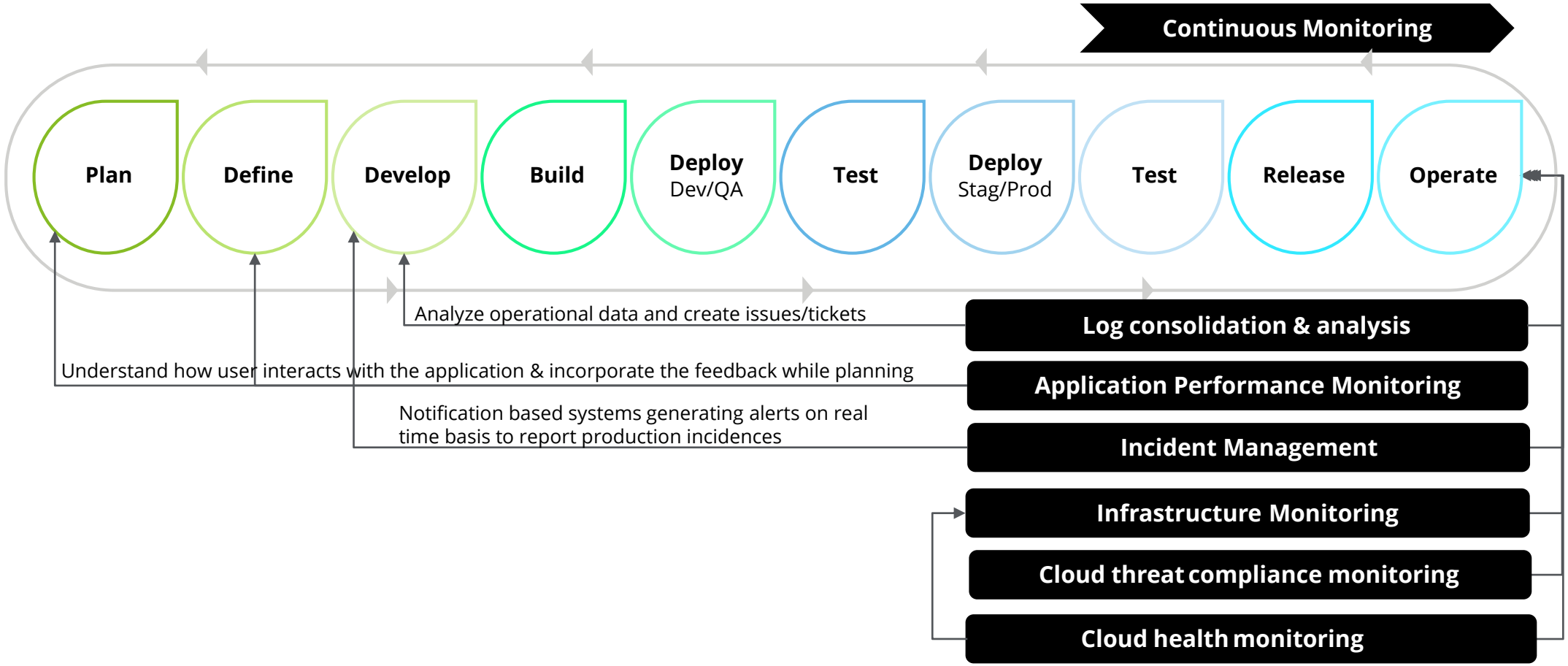
The following table provides a list of categories and common features used to evaluate each of the leading market tools

		IBM Urban Code Deploy	XebiaLabs XL Deploy	CA Release Automation	Electric Flow Deploy
Components and Workflows	Provides ability to develop reusable components and define relationship among them	✓	✓	✓	✓
	Supports role-based security	✓	✓	✓	✓
	Provides ability to develop workflows using a graphical editor	✓	X	✓	✓
Deployment Pipeline	Supports deployment to multiple environments	✓	✓	✓	✓
	Provides ability to track and monitor deployments at component level	✓	✓	✓	✓
	Supports deployment of both code and configurations	✓	✓	✓	✓
Automation	Supports auto-promotion of deployments to higher or multiple environments upon successful deployments	✓	✓	✓	✓
	Supports Release management capabilities	✓	✓	X	X
Integration	Can be integrated with all major Software Build tools including Jenkins, Atlassian Bamboo and TeamCity	✓	✓	✓	✓
	Provides out of the box mechanism for authentication and authorization, integrates with LDAP, AD and has single sign of capability	✓	✓	✓	✓
	Can be integrated with other software deployment tools including Chef, Docker and VMWare	✓	✓	✓	✓
	Can be integrated with all major project management tools including HP ALM	X	X	✓	✓

Legend
✓ = supported
✖ = not supported

Continuous Monitoring

Continuous Monitoring is about assessing the behavior and KPIs of an application through the delivery pipeline with the intent of detecting deviations early and providing stakeholders visibility into how well the software system is functioning to serve the business needs.



Wrapping security around Continuous Monitoring

Facilitate the on-going calibration of operations to business and risk priorities and drive continuous improvement through shared ticketing processes and recursive feedback loops.

Monitor

Continuous process to detect threats and issues

Development

- Capture and monitor telemetry
- Honeypot to counteract unauthorized access to systems
- Incorporate threat intel
- Continuous scanning and compliance operations

DevOps Toolchain

- Establish normal usage patterns
- Capture and monitor telemetry
- Evaluate toolchain NetFlow
- Integrated Case Management Tools

Infrastructure

- Establish normal usage patterns
- Capture and monitor telemetry
- Monitor/terminate deviations to baselines

Quick Nuggets

- Monitor for opportunities to enhance operational efficiencies and avoid repeated errors
- Deployment and WIP monitoring
- Enrich/calibrate security testing to focus on value-add feedback (e.g., false positive removal)
- Evaluate application and user security telemetry
- Establish shared prioritization and integration with enterprise incident management process
- Establish process to manage a security breach and Isolation/mitigation action

Core Capabilities

#	Leading Practice	Key Considerations	Key Benefits
1	Monitor Environment Health & Resource Consumption Patterns	▪ Tools like AppDynamics, Anturis, and New Relic provide advanced server monitoring and application performance monitoring capabilities. Ensure monitoring alerts are in place. ▪ Analyze patterns of resource consumption for useful insights (e.g., Sawtooth pattern for memory consumption: the sudden drop in used memory is a candidate symptom for a memory leak).	▪ Allows instant root cause analysis for any application issues ▪ Triggers automatic alerts when thresholds are exceeded
2	Recording of Commands Run, Authorizations and Outputs	▪ Record all the commands executed via the console in the audit log. It can include the following: 1) source file script executions; 2) other administrative operations; 3) login attempts by admin/privileged users; and 4) resources accessed by users with admin/privileged access. ▪ Audit log any access authorization changes (add, update, or revokes for user or system accounts). ▪ Maintain audit logs in a centralized, secure, and tamper-proof space.	▪ Elevated access operations audits provide useful information in the event of security issues/exposures, or compliance requirements ▪ Recording of manually executed commands not tracked in source control can provide useful information during the investigation of any subsequent errors/defects
3	Warning-, Error-, and Fatal-Level Logs	▪ To validate that a feature operates as designed, it should be instrumented to generate sufficient production telemetry. Capture these types of logs: 1) warning-level logs capture conditions that could become an error (e.g., database call taking longer than usual): 2) error-level logs focus on error conditions (e.g., API call failure); and 3) fatal-level logs tell us when we must terminate (e.g., network daemon cannot bind a network socket). ▪ Retention of log data should be sufficient to meet business and regulatory compliance needs. ▪ Maintain aggregated logs of business events, application/service logs, and environment/infrastructure logs across application/services. Consolidated logs and access to tools with ability to analyze logs to provide insights into system behavior to anticipate issues should be easily accessible to stakeholders.	▪ Allows developers to diagnose problems on their workstations and Ops to diagnose a production problem ▪ Allows Information security and auditors to review logs to confirm effectiveness of a particular control
4	Incidence Response	▪ Developers should share production incident triage responsibilities with operations staff. Conduct postmortems after incidents and track action items to avoid re-occurrence. ▪ Periodically rehearse failure scenarios with development and operations personnel. ▪ Production incidents caused by code changes and infrastructure changes should be minimal. Recovery of service after a production incident should be immediate. Failures experienced during or after releases should trigger automated rollback of the deployment.	▪ Improves the testability of the solution before a change set is released to production ▪ Quick recovery post failure

Continuous Monitoring: Leading Practices (2 of 3)

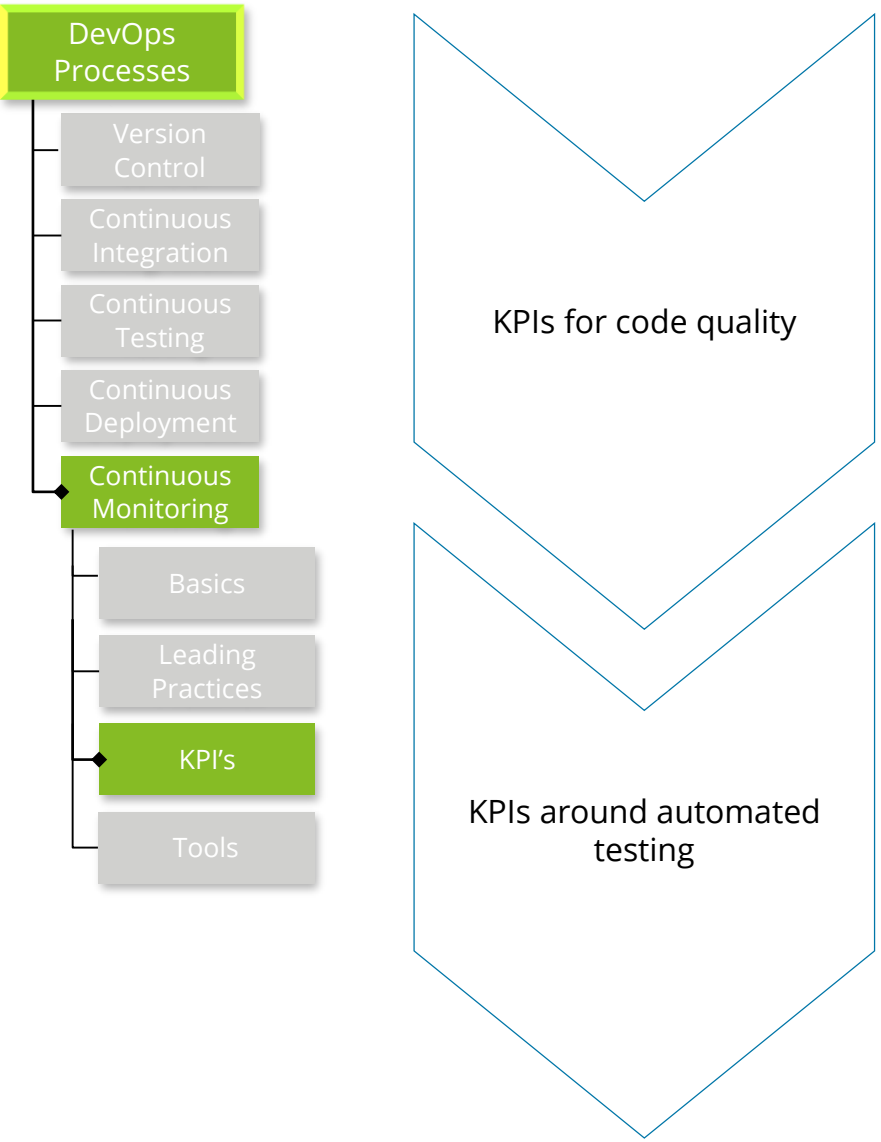
DevOps Processes and capabilities across the value stream

Advanced Capabilities

#	Leading Practice	Key Considerations	Key Benefits
5	User-Driven or System-Specific Action Logs	▪ Use action logs to track specific actions performed by various users (e.g., if a user opens a file from the library, when a user logs in, etc.) and system events (e.g., batch job runs). ▪ Correlate collected action logs with system monitoring information based on the timestamp to derive insights (e.g., system performance bottlenecks).	▪ Provides additional information to supplement issue investigation ▪ Supports A/B testing model by providing insights into how end users interact with the system ▪ Identifies opportunities for simplifying user interface/navigations
6	Continuous Monitoring of Business, App, and Infrastructure Metrics	▪ To make better decisions and anticipate problems, capture metrics continuously at all levels (i.e., business, application, infrastructure). The following telemetry dashboards are available for production and some pre-production environments: 1) At the business level, capture metrics like no. of sales transactions, average revenue, churn rate, A/B testing results; 2) At the application level, include transaction times, user response time, application faults, client-side (front-end) events and performance data, application/service transaction times, user response times, errors, etc.; 3) At the infrastructure level, capture web server traffic, CPU load, disk usage, deployment pipeline information and time, etc.	▪ Allows product manager to track business outcomes, feature usage, or conversion rates ▪ Provides the ability to see the health of everything the service relies upon using data and facts instead of rumors and assumptions
7	Statistical Analysis of Telemetry for Problem Detection	▪ Telemetry data is publicly accessible and displayed if sensitive data can be redacted. Sufficient production telemetry data allows statistical analysis that can help proactively find and fix problems before customers are affected. Use the anomaly/outlier detection technique, which involves identifying events that do not conform to an expected pattern or norm. ▪ Means and standard deviations (for Gaussian distribution): Create a filter by calculating mean and acceptable standard deviation to detect what constitutes "different from the norm". ▪ Smoothing (for time series): This involves using moving averages (or rolling averages), which transform our data by averaging each point with all the other data within our sliding window, thereby smoothing short-term fluctuations and highlighting longer-term trends or cycles. ▪ Other techniques include Fast Fourier Transforms, which is widely used in image processing, and Kolmogorov–Smirnov, which is used to find similarities or differences in periodic/seasonal data.	▪ Analyzes production telemetry to find and fix problems causing catastrophic problems earlier than ever before ▪ Finds ever-weaker failure signals that can be acted upon to create an ever safer system of work

Security Capabilities

#	Leading Practice	Key Considerations
8	Continuous security monitoring and logging	• Automates the collection, indexing and alerting of real-time machinecritical data • SIEM tool helps with correlating the events to identify security incidents andalert the security teams
9	Web Application Firewall (WAF)	• WAF that protects web applications and sites from both known and unknownattacks, including application-layer and zero-daythreats
10	Cloud threatcompliance monitoring	• To protect public cloud IaaS & PaaS resources, a focused cloud threat defense strategy is required. Cloud threat tools help us continuously monitor IaaS and PaaS for security vulnerabilities and compliance
11	Cloud healthmonitoring	• Enables continuous monitoring of performance and up-time of cloudinfrastructure hosting applications • Provides visibility into cost, configuration, usage, performance, andsecurity
12	Infrastructure vulnerability scanning	• Identifies security vulnerabilities in the underlying infrastructure including network, OS ("operating system")vulnerabilities



Code quality measures come in a variety of different forms and can be monitored at different points in the continuous deployment cycle:

- **Technical debt**: Static code analysis help developers by providing quick feedback on code quality issues related programming practices, performance, complexity, data flow, and component/ pattern reuse ratios and avoiding more costly fixes later. Catching a problem in code during the early stages of development reduces the technical debt (deferred maintenance work items) associated with application and helps with maintenance of code in long run. As part of Code analysis monitor technical debt and use code scanning tools to identify, track and resolve structural problems.
- **Code health**: Build verification tests sometimes also called as "smoke test", are small set of tests that can run quickly to determine that the code executed as expected and is ready for further testing. Build verification tests are fundamentally used to monitor code health.
- **Unit test coverage**: Each component within an application needs to work as intended. Unit test helps assess component quality in every build triggered by Continuous Integration process. Unit tests should be comprehensive and cover the common code paths, unit test coverage close to 90% should be targeted.

Once the code meets basic quality measures, subject it to greater scrutiny.

- **Functional feedback**: API-based testing provides early functional feedback. API-based testing gives early insight into whether the application as whole is working correctly. Since API-based tests are executed every time code is committed, the application is much more thoroughly tested.
- **Application security, performance, and scalability**: Early security, performance, and scalability testing uncovers flaws before they cause problems. One such performance and scalability example is defining architecture–related metrics. For example number of SQL calls made by different services with a higher number of round trips over the services indicate performance issue.
- **Successful automation rate**: Measures the increase in the percentage of automated test cases.
- **Reduced testing cycle times**: Measures how quickly defects are being caught during the test execution cycle.
- **Reduced defects in production**: Measures the number of defects caught before production.
- **Defect leak**: The defects identified in next stage post validation in previous cycle (e.g., the number of defects which were not captured by the pipeline and were identified by end-user post deployment).

Release-related KPIs

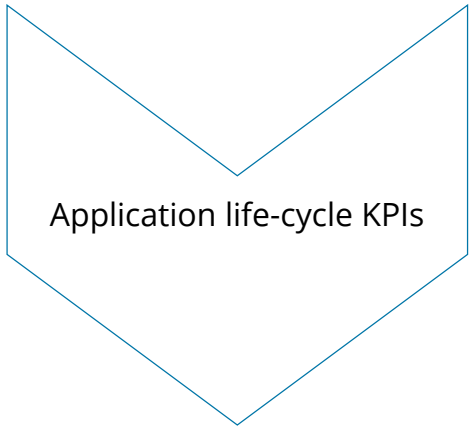
Metrics related to Release helps to measure the health of the deployment process and provides leading indicators of application stability.

- **Deployment frequency**: Deployment frequencies helps gain insights into delivery bottlenecks. Frequent deployments are desirable but consistency counts too. Deployment frequency helps highlight which processes need attention to remove inconsistencies or to automate error prone manual activities.
- **Deployment-time trends**: These trends can help highlight release abnormalities. For example, a short release cycle is desirable but huge variability might indicate that teams are achieving speed with heroic effort that cannot be sustained. A long deployment times indicate that release processes are complex, release size is too big and release might be involved in lot of manual activities.
- **Failed-deployment trends**: Manual processes are prone to error which results into deployment failures, however automation too can be prone for failure. Hence root cause analysis of failed –deployments helps teams understand the source of these failures. Deployment failures may also indicate configurations have drifted between development, test and production environments. They can also indicate problems with components shared with other applications.
- **Mean time to recover**: -It highlights how long it takes the team to recover from an issue. This is considered a true indicator of how good the team is at handling change. Spikes in mean time to recover are ok for complex issues which the team has never encountered before, but the overall trend for this metric should decrease over time.

KPIs for Application Performance Monitoring

Application Performance Monitoring provides insight into customer experience and application stability.

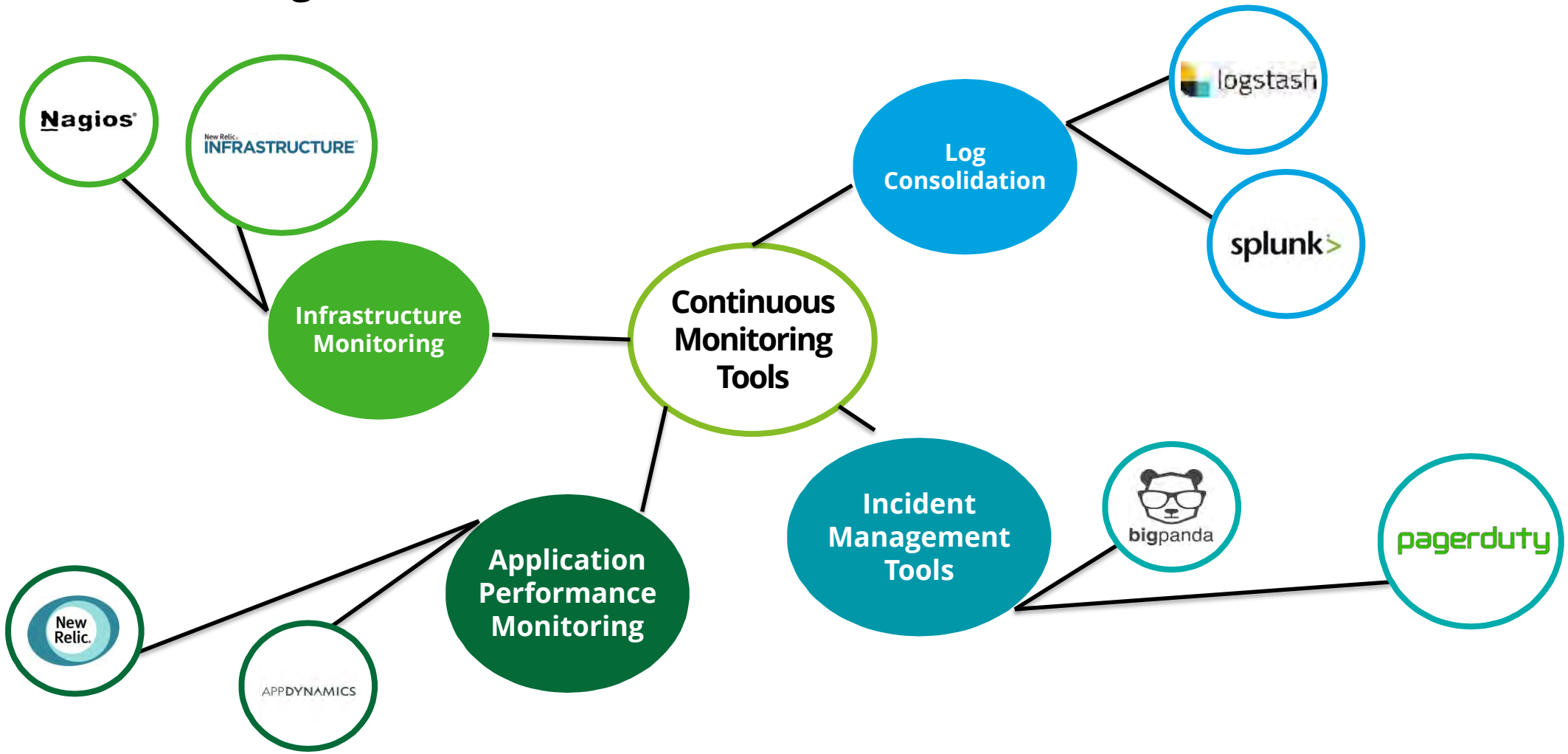
- **Feature usage**: To understand a more complete picture of user's experience, it's important to know what features of the application are actually being used the most. This information helps to better understand users, and reduce waste by shifting testing efforts more on the features being used the most.
- **Business outcomes achieved**: End users use applications to achieve business outcomes. Application developers can leverage tactical data such as user clicks, page usage, and completed transactions to understand why an outcome was good or bad. Further monitoring the business actions taken by user, needs to go all the way through from the front end to the back end.
- **Production incident statisti**cs: Mean time-to-repair (MTTR) helps track responsiveness to probable negative events, while trends on the incidents rates helps to spot increasing instabilities that need to be addressed.



Application life-cycle metrics measures the capacity to respond to change and deliver business value.

- **Monitor change request trends**: More the number of change requests indicate of more misinterpretation of requirements. This can be reduces by applying design practices to improve how to understand requirements.
- **Monitor defect count**: Defect count measures a failure to build right things. Root causes can vary. Regardless of the root cause, defects represent process failure to catch critical problems before they reach end-user.
- **Change Failure Rate**: The metrics determines the percentage of changes that results either in degraded service or need to be updated fixed with hotfixes, patches etc.
- **Lead time for changes**: This metric measures the time it takes for a code change to make it to production i.e. from code commit to code successfully running in production.

Market Leading Tools*



*** Note :- For more details on tools and features for Continuous Monitoring please refer Continuous Monitoring KPI Guideline**

DevOps transformation begins with an understanding of current state capabilities and identifying areas for improvement

Deloitte. DevOps Maturity Assessment

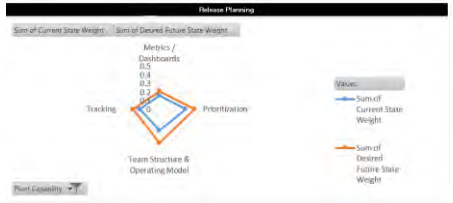
What is it?

Deloitte's DevOps Maturity Assessment (DDMA) is an extensive questionnaire for assessing current state against desired future state maturity of DevOps capabilities across the DevOps domains: from Release Planning to Continuous Deployment and Monitoring.

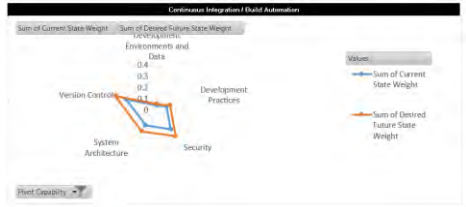
Benefits

- 180 questions along each of the DevOps domains (Release Planning, Continuous Development, Continuous Integration, Continuous Testing, Continuous Deployment, Continuous Monitoring
- Assesses maturity against desired future state
- Identifies areas for capability improvement

Sample Output



Release Planning Maturity



Continuous Integration Maturity

Co-create Transformation Model: Current State Assessment (2 of 4)

DevOps Transformation enables the seamless adoption of DevOps across various levels, i.e., project, product, and organization levels.

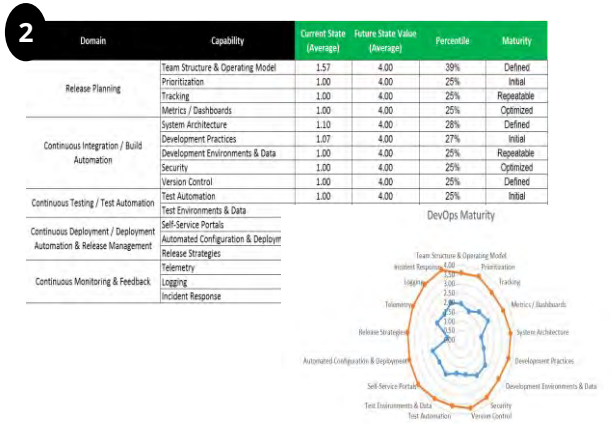
Deloitte's DevOps Maturity Assessment Model output will give a maturity "ranking" for each capability; this maturity ranking will be used along with transformation guidelines to assist in creating a high-level transformation roadmap

1

Maturity Assessment & Interviews

- Customer chooses Deloitte DevOps Maturity Assessment
- Questionnaire sent to desired recipients
- Supplement questionnaire results with interviews of necessary individuals

2



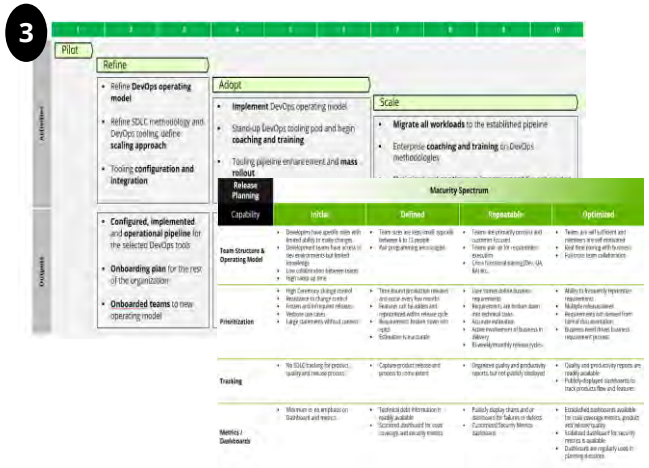
Domain	Capability	Current State (Average)	Future State Value (Average)	Percentile	Maturity
Release Planning	Team Structure & Operating Model	1.57	4.00	39%	Defined
	Prioritization	1.00	4.00	25%	Initial
	Tracking	1.00	4.00	25%	Repeatable
	Metrics / Dashboards	1.00	4.00	25%	Optimized
Continuous Integration / Build Automation	System Architecture	1.10	4.00	28%	Defined
	Development Practices	1.07	4.00	27%	Initial
	Development Environments & Data	1.00	4.00	25%	Repeatable
	Security	1.00	4.00	25%	Optimized
	Version Control	1.00	4.00	25%	Defined
Continuous Testing / Test Automation	Test Automation	1.00	4.00	25%	Initial
	Test Environments & Data				
Continuous Deployment / Deployment Automation & Release Management	Self-Service Portals				
	Automated Configuration & Deployment				
	Release Strategies				
Continuous Monitoring & Feedback	Telemetry				
	Logging				
	Incident Response				

DevOps Maturity

Maturity Analysis & Reporting

- Analyze maturity assessment and interview results
- Determine maturity ratings for each identified capability

3



Transformation Roadmap Design

- Identify activities to mature capabilities
- Schedule activities on transformation roadmap along Refine / Adopt / Scale timeframe

Co-create Transformation Model: Current State Assessment (3 of 4)

DevOps Transformation enables the seamless adoption of DevOps across various levels, i.e., project, product, and organization levels.

The DevOps Maturity Assessments is accompanied by interviews in order to address gaps and answer questions resulting from analyzing results of the DevOps Maturity Questionnaire

1

Completed Assessment Questionnaire

Pre-Interview

- Key Stakeholders Identified
- Head's Up Expectations Set
- Analysis of DevOps Maturity Assessment Questionnaire Results to identify interview themes
- Interviews Scheduled on Calendars

2

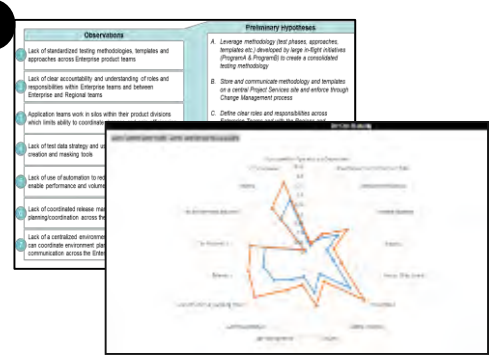
Interview Themes

- Typical duration 30 minutes
- Focus, by Interviewee Team
  - Planning
  - Development and Testing
  - Deployment and Release management
- Establish follow-up as necessary for additional info or drill-down into key subject areas

Interviewees

- Product Owners / Business Analysts
- Development Leads
- Build Engineers
- Release Coordinators
- Testing Leads

3



Key Outcomes

- Key Interview Findings (Strengths, Challenges)
- Preliminary Maturity Ratings
- Combine interview findings with Assessment questionnaire results

Deloitte DevOps Maturity Assessment Model

The Deloitte DevOps Maturity assessment assesses an organization across 5 capability domains:

Capabilities Assessed

Release Planning	Continuous Integration / Build Automation	Continuous Testing / Test Automation	Continuous Deployment & Release Management	Continuous Monitoring & Feedback
Team Structure & Operating Model	System Architecture	Test Automation	Self-Service Portals	Telemetry
Prioritization	Development Practices	Test Environments & Data	Automated Configuration & Deployment	Logging
Tracking	Development Environments & Data		Release Strategies	Incident Response
Metrics / Dashboards	Security			
	Version Control			

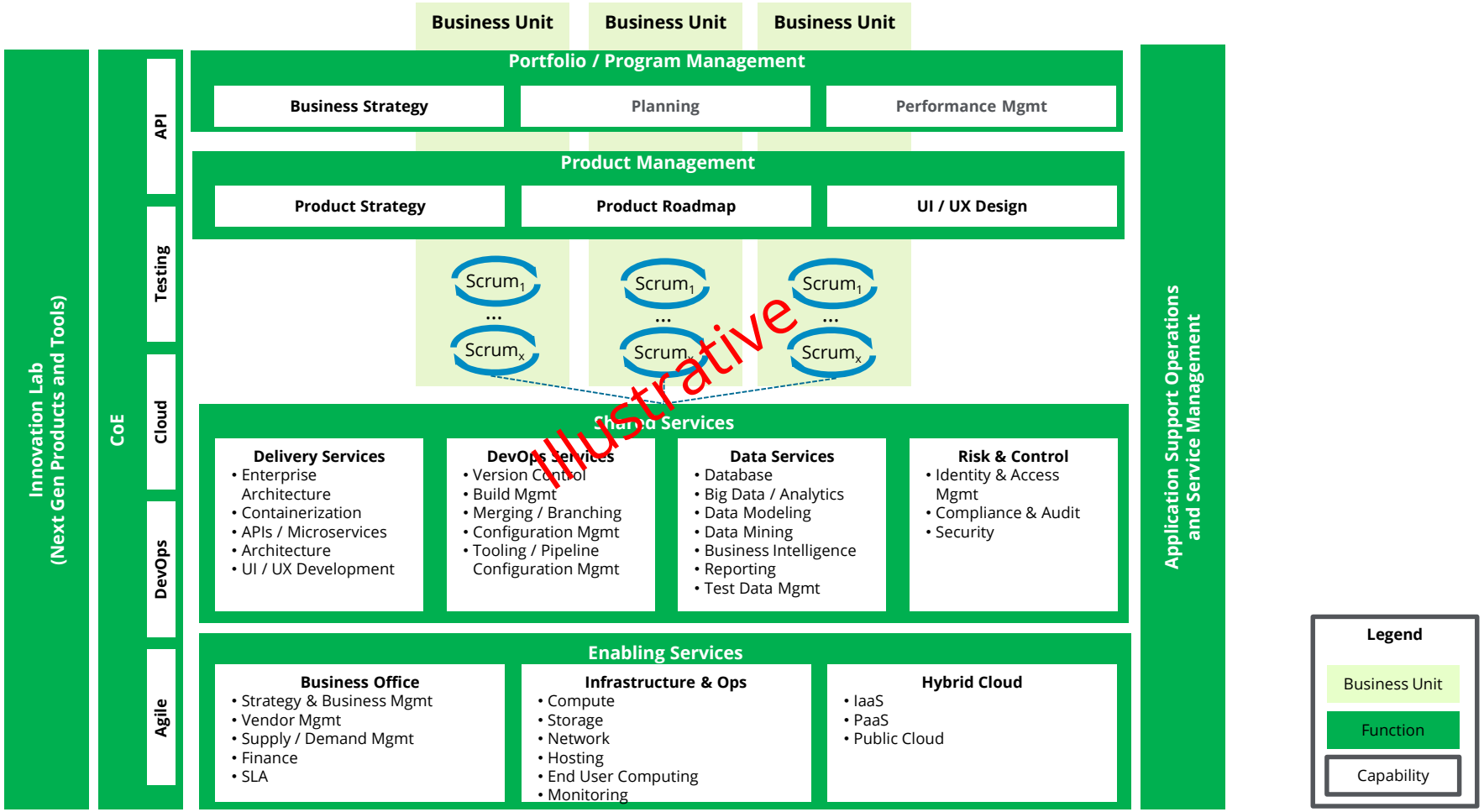
DevOps Operating Model Guiding Principles

Technology integrations and service operations connect seamlessly to create a DevOps organization that can show quick successes and agile, rapid responsiveness to their customer's application development needs

DevOps First	▪ Extend Agile philosophies by embracing constant testing and delivery through iterative changes ▪ Empower "two pizza teams" to continuously integrate and deliver change ▪ Culture of multiple releases and rapidly deploying code to production
Extreme Automation	▪ Zero touch operations organization ▪ Self service provisioning
Measure everything	▪ Continuous monitoring and measurement ▪ Metrics driven change
Fail Fast Experimentation	▪ Fail fast, Fail often SDLC cycle ▪ Continuous Feedback loops
Cloud First	▪ IT organization includes cloud as part of business and technology strategy and solution ▪ New development is cloud native applications

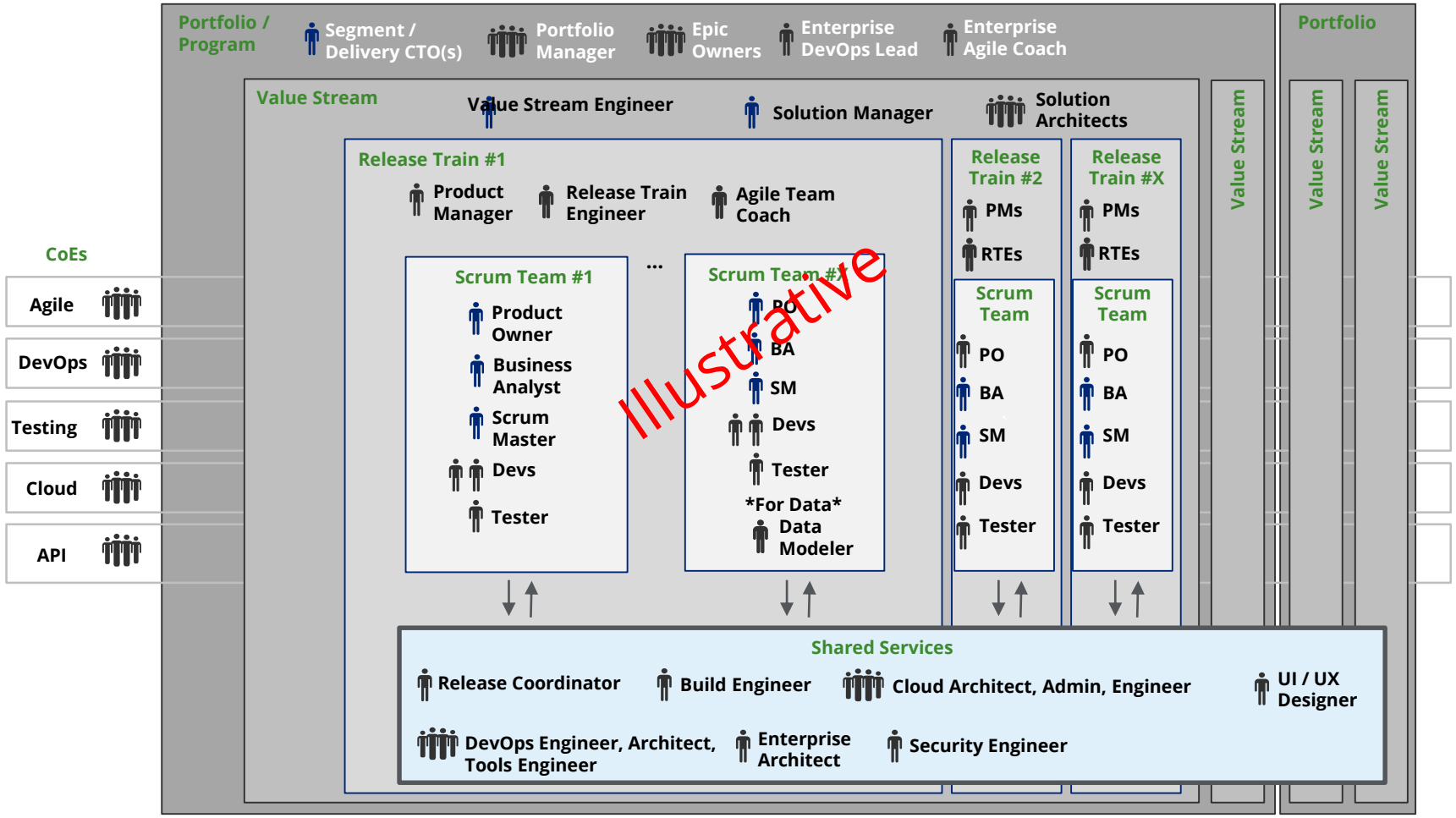
DevOps Operating Model: Capability View

This operating model represents an organization with high DevOps maturity, rapidly responding to changing market needs



Target Operating Model: Role View

The roles listed below support the operating model and focus on maximizing business value, agility, and accountability for a given portfolio and value stream

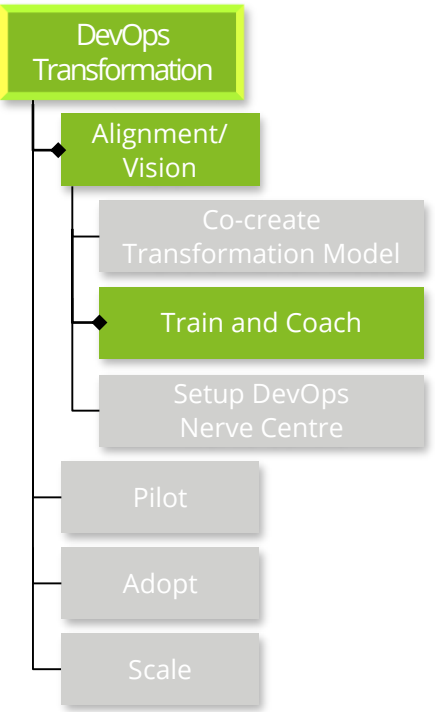


Transformation Roadmap

The high-level transformation roadmap gives a broad set of activities and outputs for each stage of a DevOps transformation; depending on the results of a maturity assessment, activities can be derived from the Transformation Guidelines

Month	1	2	3	4	5	6	7	8	9	10
	Pilot	Refine		Adopt			Scale			
Activities	▪ **DevOps** pilots	▪ Refine **DevOps operating model** ▪ Refine SDLC methodology and DevOps tooling, define **scaling approach** ▪ Tooling **configuration and integration**		▪ **Implement** DevOps operating model ▪ Stand-up DevOps tooling pod and begin **coaching and training** ▪ Tooling pipeline enhancement and **mass rollout**			▪ **Migrate all workloads** to the established pipeline ▪ Enterprise **coaching and training** on DevOps methodologies ▪ Optimized and **continuous improvement** for onboarded and existing teams			
Outputs	▪ Assessment results ▪ Transformation roadmap ▪ Initial toolchain ▪ Op model blueprint	▪ **Configured, implemented** and **operational pipeline** for selected DevOps tools ▪ **Onboarding plan** for the rest of the organization ▪ **Onboarded teams** to new operating model		▪ Configuration and deployment of pipeline **across the organization** ▪ **Trained and supported client teams** on the new development and deployment process			▪ **Supported client teams** with migrating their efforts across the enterprise ▪ **Defined CoE** to drive improvement of DevOps pipeline ▪ **Self-sustaining** DevOps teams			

Illustrative



Training and Workshop

Objective	✓ The key objective of conducting workshops is to improve the level of leadership commitment around the project and people development and to finalize the roadmap
Detailed Activities	✓ Accumulate Training Material :- Create training assets by leveraging existing material to deliver training workshops ✓ Identify the least common denominator around the understanding of DevOps ✓ Bring all stakeholders on the same page with regard to final objectives and definitions of terms ✓ Conduct workshops that will help achieve the aim of improving commitment and demonstrate leadership involvement ✓ Refine target DevOps operating model and roadmap

Alignment/Vision: Train and Coach (2 of 2)

DevOps Transformation enables the seamless adoption of DevOps across various levels, i.e., project, product, and organization levels.

DevOps Transformation
- Alignment/ Vision
  - Co-create Transformation Model
  - Train and Coach
  - Setup DevOps Nerve Centre
- Pilot
- Adopt
- Scale

The following are some sample training and workshops which can be conducted at this stage:

Training Module	Learning Objectives	Audience
Introduction to DevOps *Training*	▪ Define what DevOps is, its core principles, and goals ▪ Describe the value DevOps brings to an organization and what it means to be a "DevOps" organization ▪ Describe the difference between DevOps, agile/hybrid agile project management, and traditional waterfall SDLC project management ▪ Explain the DevOps maturity model and transformation model ▪ Describe DevOps best practices, goals, and myths ▪ Describe DevOps enabling tools	▪ Management / Executives ▪ Scrum Master / Scrum Teams ▪ Product Owner
Building a Delivery Pipeline *Workshop*	▪ Describe end-to-end delivery pipeline and how to integrate with existing toolchain ▪ Describe toolchain configuration best practices ▪ Explain integrations and define connections with existing pipeline ▪ Describe the different stages of delivery pipeline: Build, Execute Unit Tests, Static Code Analysis, Security and Vulnerability Testing, Packaging and Artifacts Management, Environment Provisioning, Deployment	▪ DevOps / Release Engineer ▪ System Admin ▪ Tools Administrator ▪ Build Architect / Engineer
DevOps Discovery *Workshop*	▪ Assessment of business needs, pain points, and DevOps maturity ▪ Review results of DevOps Maturity Assessment ▪ Identification of how DevOps processes and tools can help to address problems	▪ Management / Executives ▪ Scrum Master / Scrum Teams ▪ Product Owner

DevOps Nerve Center

Objective	✓ A DevOps COE, often referred to as the DevOps Nerve Center, is chartered in a 'pre-pilot' stage and put in place in the 'adopt' stage. It aims to share, support and advocate DevOps across the organization
Detailed Activities	✓ Form DevOps Nerve Center team with line of sight to • Project Leadership • Business Leadership • Development, Testing • Infrastructure, Operations • Enterprise Architecture • Security/Compliance ✓ The DevOps Nerve Center is responsible for overseeing and carrying out all activities related to conceptualization and identifying projects for the pilot.

Pilot: Preparation for Pilot

DevOps Transformation enables the seamless adoption of DevOps across various levels, i.e., project, product, and organization levels.

DevOps Transformation
- Alignment/ Vision
- Pilot
  - Preparation for Pilot
  - Execute Pilot
- Adopt
- Scale

Identify Pilot Candidate

Develop a short list of projects that would be good candidates for a pilot project and select the best one. Consider critical, mid-size projects (based on duration and team size) with low risk and integration complexity.

The project must also have enough business stakeholder focus for the project team to work well to ensure its success.

Define the Integrated Processes

Based on the team's process maturity, determine which set of integrations to adopt first to make progress along the process agility curve. Elements of the process agility curve are described below:

- Source Code Control: Single source code repository.
- Build Automation: Compile and build application without much human intervention.
- Continuous Integration: Developers regularly check in code changes to a shared repository, which triggers the build and test execution.
- Continuous Testing and Test Automation: Execute unit, integration, service, security, and performance tests without much human intervention.
- Continuous Deployment: Packaging, deploying, and post-deployment testing occurs without much human intervention.
- Continuous Delivery: Every change to the system can be released at the push of a button.
- Continuous Monitoring: Gauge the success or failure of the pilot by continuously monitoring DevOps processes.

Select appropriate tools*

The tool selection depends on the integrated processes defined, team familiarity with the platform and the vendor preference. The tools should help enable the process being implemented and assists with the implementation effort.

The tool selection needs to be a balanced one and needs to support organization wide adoption at a later time but not impede the pilot team adoption. The tool selection at pilot stage only involves the tools that are applicable to the pilot project so that the team can focus on tools of use only.

*** Note :- For more details on market leading DevOps tools, features please visit DevOps Point of View**

Pilot: Execute Pilot

DevOps Transformation enables the seamless adoption of DevOps across various levels, i.e., project, product, and organization levels.

DevOps Transformation
- Alignment/ Vision
- Pilot
  - Preparation for Pilot
  - Execute Pilot
- Adopt
- Scale

Establish Infrastructure and Tools

- Integrate processes, minimizing manual handoffs.
- Set up the tool configuration and integrations to create a seamless journey of the code artifacts throughout the delivery cycle.
- Automate the complex workflows required for code change movement between development, test, and production environments by removing manual interaction and avoidable errors.
- Make sure to address considerations for these key components:
  - Source Code and Build System
  - Issue Tracking and Monitoring System
  - Communications Systems
  - Deployment and Rollback
  - Infrastructure Provisioning

Configure Pipeline

To configure the DevOps pipeline, capture the following aspects of the key pipeline components:

- Source control management (SCM): This system should capture software changes, triggering the remaining automated DevOps build, test, and deploy activities.
- Project management tool: This tool should be a central place for tracking requirements, current issues, and timelines. The tool must allow project visibility across the organization and provide strong reporting capabilities. It also needs to integrate with other tools to provide end-to-end traceability.
- Build system: This system must support continuous integration (i.e., building the software, running unit and validation tests, deploying to the environments, and performing any other automated tests needed). Integration with SCM and the validation system is key. Support for workflows to deploy to environments is an added advantage.
- Collaboration tools: Collaboration can be implemented using one or more systems (e.g., email, wikis, and a real-time chat system), but they must support real-time notifications or integration with the user's tools so that status is visible within the user environment (i.e., avoid the need to go to a predefined place or launch an app to see status).
- Deployment systems: Deployment systems should integrate with the build system so that code artifacts can be deployed and promoted to various environments based defined quality gates. Support for workflow and application modeling is a key tool capability.
- Infrastructure: The infrastructure should be standardized (i.e., definition, configuration, and build out also need to be standardized to avoid manual errors). Using tools that facilitate "infrastructure as code" will significantly ease deployments.
- Monitoring systems: This system should have the ability to track all project-related systems. It needs to be integrated with the issue tracking and communications systems so that notifications can be sent out in real time with access to performance monitors.

To achieve automation across the delivery cycle, the team should study and consider the needs of the entire life cycle of change in the project—from inception through deployment. This will result in a superior product due to the increased focus on implementation details and operational realities.

In addition to tool integration, collaboration and visibility is key. These structured interactions should ensure that developers, quality assurance staff, managers, and external stakeholders receive continuous, real-time information about project status, which is beneficial to quicker project agility.

Adopt: Leverage Learnings

DevOps Transformation enables the seamless adoption of DevOps across various levels, i.e., project, product, and organization levels.

Review DevOps pilot feedback and lessons learned

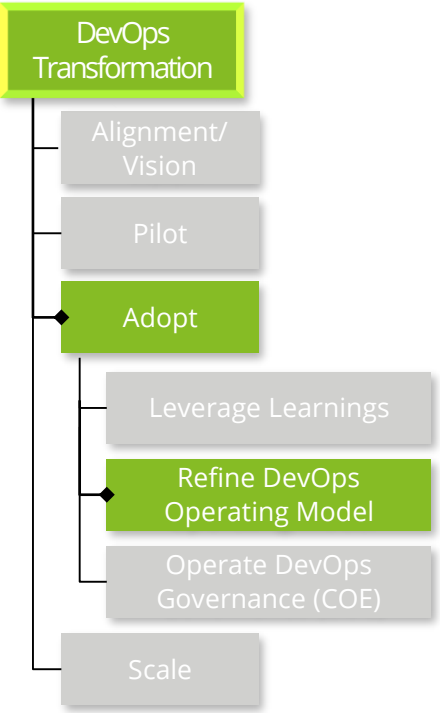
- Engage all stakeholders and identify process improvement areas.
- Conduct a session with all stakeholders to understand pain points.
- Review the Stakeholder feedback for the DevOps pilot.

Adapt

- Review the process set of the DevOps pilot.
- Tune the tools and processes to better suit the DevOps project adoption candidates.
- Identify and implement any toolset changes that are required.
- Document the generic criteria, rules, and standard practices for all DevOps project adoption candidates, providing enough flexibility for any adoption candidate to customize according to the business need. Stipulate a generic boundary within which the customizations can happen.

Adopt: Refine DevOps Operating Model

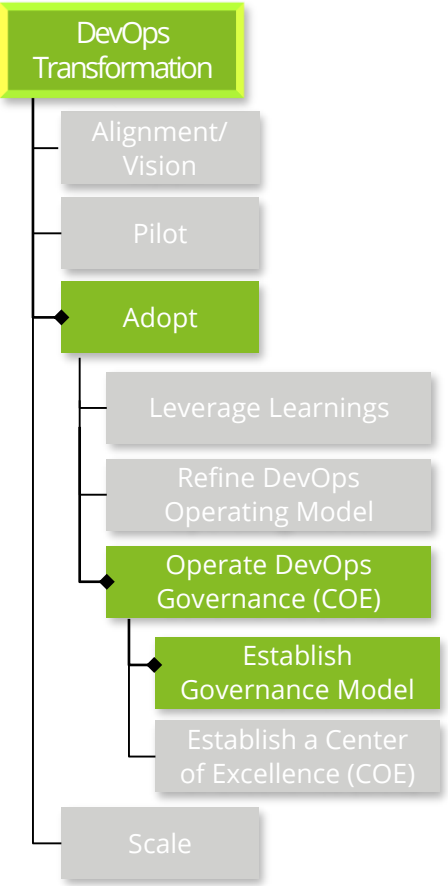
DevOps Transformation enables the seamless adoption of DevOps across various levels, i.e., project, product, and organization levels.



Objective	✓ To refine the target DevOps Operating Model to adopt proper project candidates (based on learnings from the Pilot)
Detailed Activities	✓ Identify similarities and differences of the Organization Operating Model adoption candidates: The selected adoption candidates may have similarities and differences with regard to people, the tools used, and the business objectives to achieve. Define a criterion to determine the current state and then derive the Organization operating model. The organization operating model should optimize end to end processes to help deliver value. ✓ Remember to apply pilot project learnings: Use the pilot project to determine what works for the organization and to make course corrections when failures occur along the journey. Use the project as a catalyst or pivot to demonstrate values. Apply the lessons and knowledge learned to bigger projects. ✓ Promote the Right-Speed IT approach: IT organizations must try to adopt a delivery model that balances high-torque enterprise operations and high-speed innovation. Understand the dimensions of Right-Speed IT transformation to help set up a successful transformation program. Based on the nature of the application, one or more transition steps may be needed to determine the right speed of delivery as a whole.

Operate DevOps Governance (COE): Establish Governance Model

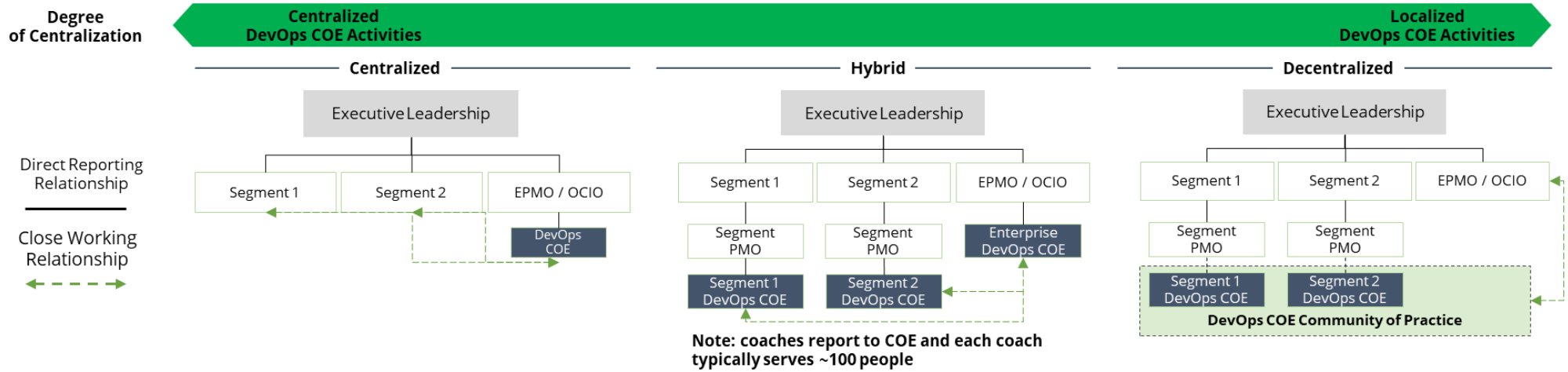
DevOps Transformation enables the seamless adoption of DevOps across various levels, i.e., project, product, and organization levels.



Objective	✓ Establish DevOps Governance to define the process controls in order to ensure that various projects and organization elements are following the checks and balances.
Detailed Activities	✓ Ensure that Governance oversees the rollout of the target organization operating models to teams either as incremental or one time rollouts. ✓ Ensure that Governance identifies and resolves impediments that hinder the successful rollout of the models and ensures consistent adoption across the organization. ✓ Governance should try to validate against agreed-upon metrics e.g., rate of adoption. ✓ Ensure the adherence to audit, compliance, and controls requirements that affect defined models. ✓ Consider gathering data-driven feedback to understand performance and identify focus areas that can be improved. ✓ Try leveraging experiences and collating various assets from completed projects so the COE can help with the organization-wide scaling.

Establishing a Center of Excellence (COE)

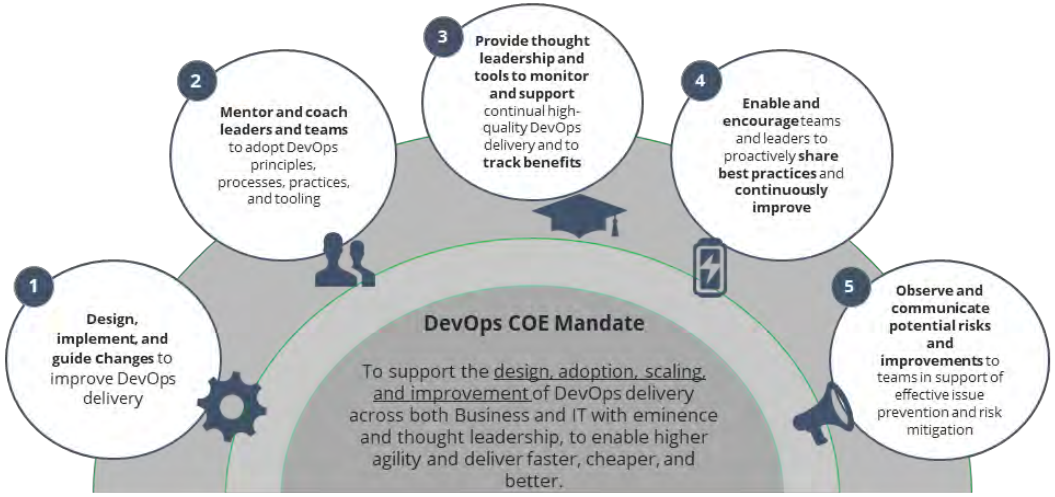
There are three ways to organize an enterprise-level DevOps COE.



	Centralized	Hybrid	Decentralized
Pros	• Higher service quality, consistency, and standardization • More effective utilization and cost management because resources are deployed across segments on an as-needed basis	• A certain degree of standardization is enabled because the Enterprise COE provides basic enterprise-wide service definition and standards • With guidance from the enterprise, segments are empowered to customize standards, processes, methods, and tools based on their needs	• Most flexible because segments define their own standards, processes, methods, and tools • A DevOps COE Community of Practice will enable the sharing of best practices across segments
Cons	• Limited ability to provide more customized services that are more appropriate for a specific segment • Difficult to coordinate DevOps COE activities across the enterprise with multiple segments	• Service quality may suffer if there is no close coordination with the Enterprise COE • Could be more costly to operate due to potential duplication of capabilities	• Lack of effective coordination of DevOps delivery across segments to ensure alignment • Lack of consistency in service standards and quality • Lack of transparency in degree of use of segment DevOps COE due to localized teams

Determining DevOps COE mandate, services, and role

DevOps COEs have a common mandate and set of services that are provided. However, the operating model dimensions have different options available that define the level of effectiveness the COE can directly deliver.



Services			
Change Planning and Rollout Management	**Governance and Metrics**	**Training, Coaching, and Change Management**	**Process and Tooling**
Defines the change strategy and program manages the rollout	Defines and stewards of the governance and metrics tracking	Provides training, coaching, and change management to adopters	Defines and provides support for process and tooling for DevOps

Dimensions	Typical Options
Engagement Style	**Pull-Based vs. Push-Based** • Pull: Provide pull-based services based on an opt-in style for adopters • Push: Engages based on a pre-determined rollout plan and goes all in
Service Model	**Enable vs. SWAT vs. Operate** • Enable: Sets standards and provides support through primarily coaching and communities of practice • SWAT: Has a special team that goes in and executes with adopters • Operate: Owns a part of the value chain (e.g., release)
Funding Approach	**Independent vs. Shared** • Independent: COE budget is set and funded from the top-down • Shared: COE receives consumption-based funding and adopters are expected to contribute to funding
Resourcing	**Fixed vs. Rotational** • Fixed: COE resources are assigned to their roles for a long term; new permanent job descriptions are created • Rotational: COE is sourced primarily through a rotational program and individuals are seconded for a short term

Scale: Train advanced automation topics (1 of 5)

DevOps Transformation enables the seamless adoption of DevOps across various levels, i.e., project, product, and organization levels.

Objective	✓ To train team members on CI/CD methodology, best practices, specific vendor tool training to assist with project specific usage and integrations
Detailed Activities	✓ **Training on Tools and Processes** ▪ Validate that the following DevOps foundational capabilities should be in place for successful scaling across the organization :- • Fully automated build, test and deploy process that runs continuously • Cross-functional Teams with resources from dev, test, infra and operations • Implement dynamic environment provisioning and testing/recovery • Infrastructure and Ops as Code leading to management of infrastructure and operations like a “product and platform” ✓ **Process of Collective Learning** ▪ Advocate meeting and collaboration across teams through COE forums which will help create a team culture around shared success and evaluate the organization as a whole

Scale: Train advanced automation topics (2 of 5)

DevOps Transformation enables the seamless adoption of DevOps across various levels, i.e., project, product, and organization levels.

DevOps Transformation
- Alignment/ Vision
- Pilot
- Adopt
- Scale
  - Train advanced automation topics
  - Support the new Organization Rollout
  - Enable continuous & Quantifiable improvement

The following modules can be offered to customer teams along each stage of the Adopt / Scale phase to help transform DevOps adoption enterprise-wide

Training Module	Learning Objectives	Capabilities Addressed	Audience
Release Planning *Workshop*	▪ Explain how agile and hybrid agile project management concepts apply to and are incorporated with DevOps ▪ Describe the Roles and Responsibilities of Scrum Master, Scrum Team, Product Owner, QA / Tester, Role of Developer, Release Manager, System Administrator ▪ Describe how to plan a sample Scrum with DevOps ▪ Explain optimization of delivery pipeline (e.g. collapsing stage gates between phases, automation of workflow) ▪ Define KPIs for DevOps projects and how work gets done and measured ▪ Define metrics around Product and Release Quality ▪ Explain unit tests and code coverage reports ▪ Explain Security & Vulnerability Assessment reports ▪ Explain Functional and Performance testing reports ▪ Define metrics around Technical Debt and SQALE (Software Quality Assessment Based on Lifecycle Expectations)	▪ Team Structure & Operating Model ▪ Prioritization ▪ Tracking ▪ Metrics / Dashboards	▪ Release Manager ▪ QA Lead ▪ Portfolio Owners ▪ Delivery Head ▪ Product Management

Scale: Train advanced automation topics (3 of 5)

DevOps Transformation enables the seamless adoption of DevOps across various levels, i.e., project, product, and organization levels.

DevOps Transformation
- Alignment/ Vision
- Pilot
- Adopt
- Scale
  - Train advanced automation topics
  - Support the new Organization Rollout
  - Enable continuous & Quantifiable improvement

The following modules can be offered to customer teams along each stage of the Adopt / Scale phase to help transform DevOps adoption enterprise-wide

Training Module	Learning Objectives	Capabilities Addressed	Audience
Continuous Integration: Version Control *Workshop*	▪ Define software version control ▪ Describe the benefits of using software version control ▪ Define and describe the difference between centralized systems and distributed systems ▪ Define software version control branching and merging ▪ Define software version control tags and labels ▪ Define workflow integration and continuous build ▪ Describe parallel development branching models and merging ▪ Describe Developer and Continuous Integration workflows ▪ Explain version control best practices ▪ Explain software version control stand functionality and features	▪ Version Control	▪ Scrum Team ▪ Developers ▪ Scrum Master ▪ Release/Project Manager ▪ Build/Release engineer
Continuous Integration: Build Automation *Workshop*	▪ Describe the continuous integration process and sub processes ▪ Describe the flow of control/data within continuous integration processes ▪ Describe developer DevOps workflow ▪ Define IDE integrations ▪ Explain the code integration process	▪ System Architecture ▪ Development Practices ▪ Development Environment & Data ▪ Security	▪ Developers ▪ Dev/Release lead ▪ Scrum Team ▪ Scrum Master ▪ Release/Project Manager ▪ Build/Release Engineer

Scale: Train advanced automation topics (4 of 5)

DevOps Transformation enables the seamless adoption of DevOps across various levels, i.e., project, product, and organization levels.

The following modules can be offered to customer teams along each stage of the Adopt / Scale phase to help transform DevOps adoption enterprise-wide

Training Module	Learning Objectives	Capabilities Addressed	Audience
Continuous Testing / Test Automation *Workshop*	▪ Define and describe traditional testing practices and continuous testing practices ▪ Identify and differentiate between traditional testing practices and continuous testing practices ▪ Define automated unit, functional (System, Integration), and non-functional testing (perf, code analysis, security) ▪ Define continuous testing process and sub processes ▪ Define the transition from continuous integration to continuous testing as part of the SDLC ▪ Highlight the benefits of test automation and value it brings to clients/projects ▪ Integrate automated testing processes with the continuous integration process	▪ Test Automation ▪ Test Environment & Data	▪ Scrum Team ▪ Scrum Master ▪ Release/Project Manager ▪ Build/Release Engineer ▪ Developers ▪ QA Lead ▪ QA Teams
Continuous Deployment / Deployment Automation & Release Management *Workshop*	▪ Define continuous deployment and continuous delivery ▪ Describe and identify the difference between continuous deployment and continuous delivery ▪ Define software defined infrastructure (SDI) or infrastructure as code ▪ Describe the benefits of software defined infrastructure (SDI) or infrastructure as code ▪ Explain the process of standardizing resource configurations	▪ Self-Service Portals ▪ Automated Configuration & Deployment ▪ Release Strategies	▪ Scrum Team ▪ Scrum Master ▪ Release/Project Manager ▪ Build/Release Engineer ▪ Developers ▪ Infrastructure Teams

Scale: Train advanced automation topics (5 of 5)

DevOps Transformation enables the seamless adoption of DevOps across various levels, i.e., project, product, and organization levels.

DevOps Transformation
- Alignment/ Vision
- Pilot
- Adopt
- Scale
  - Train advanced automation topics
  - Support the new Organization Rollout
  - Enable continuous & Quantifiable improvement

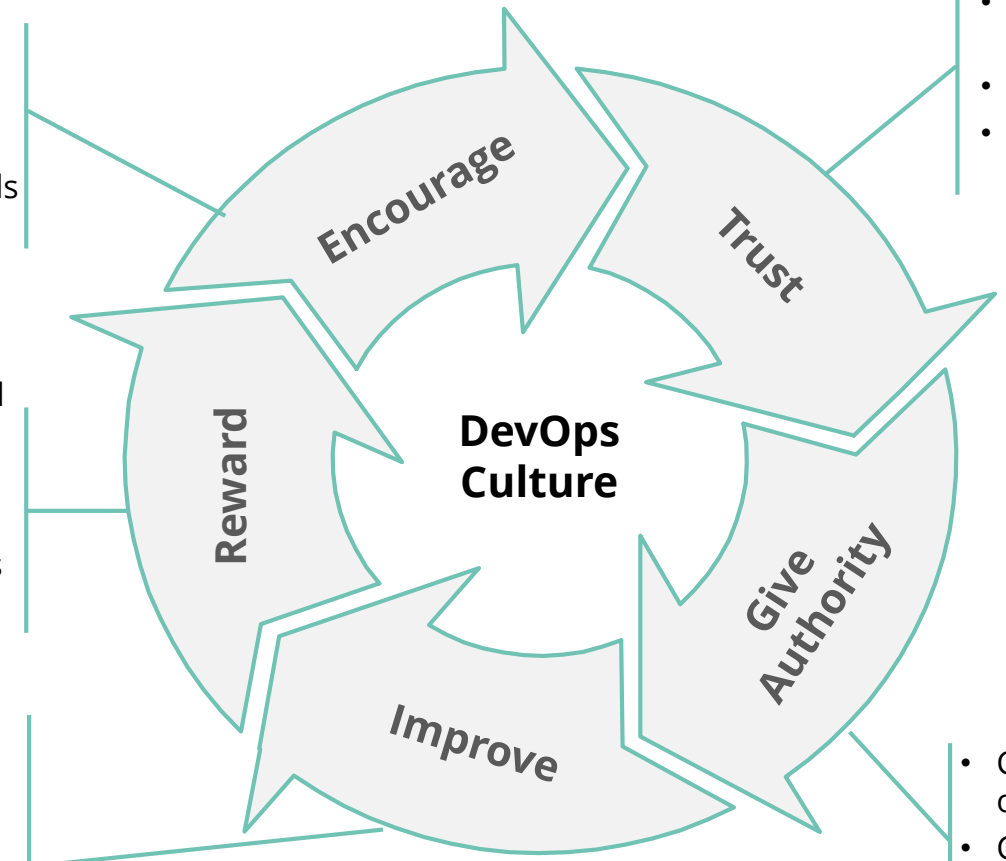
The following modules can be offered to customer teams along each stage of the Adopt / Scale phase to help transform DevOps adoption enterprise-wide

Training Module	Learning Objectives	Capabilities Addressed	Audience
Continuous Monitoring & Feedback *Workshop*	▪ Define and describe continuous monitoring ▪ Define and describe continuous feedback ▪ Describe marketplace leading continuous monitoring tools and their capabilities	▪ Telemetry ▪ Logging ▪ Incident Response	▪ Scrum Team ▪ Scrum Master ▪ Release/Project Manager ▪ Build/Release Engineer ▪ Operations Teams

Building a DevOps culture

The below components will play a crucial role in strengthening the workforce and building a DevOps culture

- Cultural change doesn't happen without top-down sponsorship and active participation
- Executive decisions usually take more time to trickle down to all levels



- Requires control functions to trust that product teams
- Trust needs to be earned
- Teams collaborate and demonstrate success

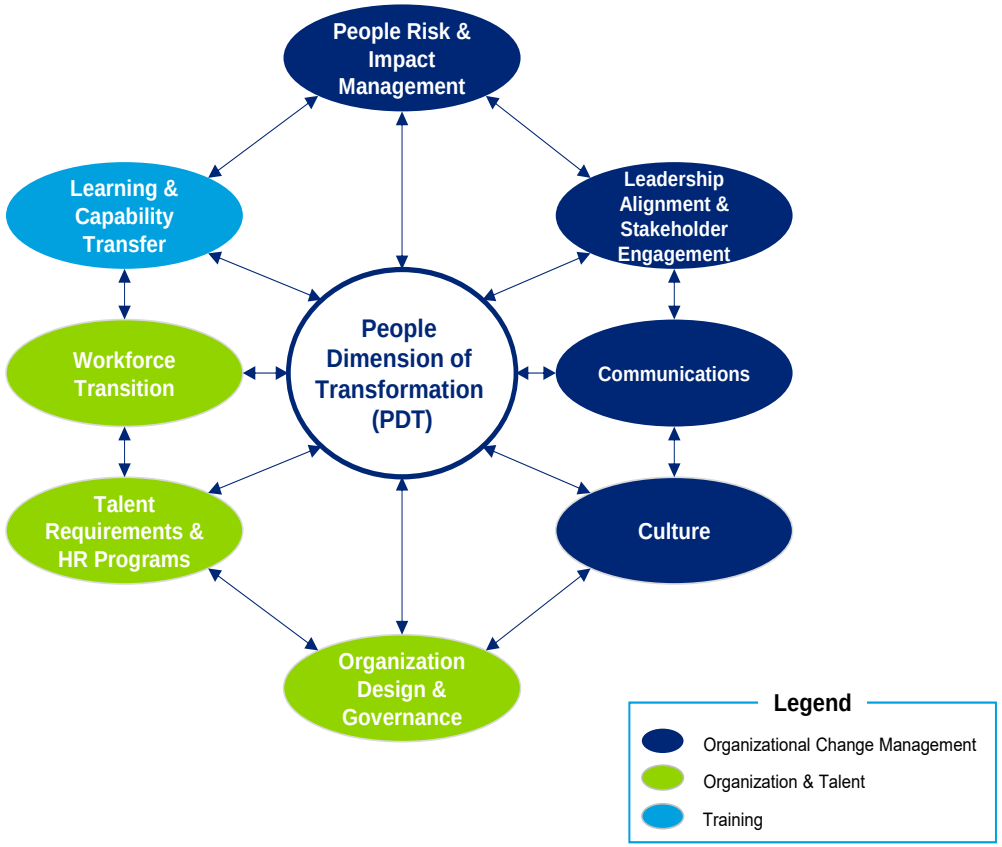
- People are measured and rewarded for the right things
- Materially improve the odds of success
- Successful large-scale change starts small

- Hunger to improve
- Changing mind-sets from "Achieving perfection" to "Good enough"
- Putting in place flexible systems

- Control responsibilities formerly owned by other functions
- Control must be designed into the process right from the start
- Reimagining how controls are implemented

Delivering Cultural Change

Managing the change with the right leadership, behaviors and targeted interventions is critical in ensuring the new DevOps Operating model is effective. People Dimension of Transformation (PDT)* framework identifies eight key components that are needed to consider as part of change management and organization enablement to adopt DevOps



Quick Nuggets

- People Dimension of Transformation (PDT)* framework provides an integrated, targeted, and repeatable approach to planning, developing, deploying and continuously improving upon change solutions associated with organization transformation efforts
- The PDT framework is organized into three focus areas: Organizational Change Management, Organization & Talent, and Training.
- It assesses the current environment, outlines the appropriate change strategy to achieve business objectives, and delivers the interventions required to successfully deliver on the change.

Note :- For more details on PDT framework please refer O&C: People Dimension of Transformation

Delivering Cultural Change

PDT component		Main objectives
Organizational Change Management	**Leadership alignment & stakeholder engagement**	• Empower and align all leaders to be committed to driving the transformation from the top • Plan for sustained transformation by building deep, personal commitment at every level of the organization • Engage all stakeholder types, as effective transformation requires contributions and involvement from all types of players
	Communications	• Create compelling communications that convey the reason for change and the importance of the transformation • Build stakeholder support by engaging in two-way communication about the true impact of the transformation, internally and externally • Develop and distribute branded, consistent communications to internal and external stakeholders
	Culture	• Understand the key attributes of the current culture and how it is likely to be affected by transformation • Develop a desired culture, identify key drivers and develop initiatives to help shift or align from current state and avoid an organization stuck in between new ways of working and old modes of behavior • Drive alignment of behavior and desired future cultural objectives; establish the right leadership models and introduce new words and vocabulary that highlight the desired behavior
	People risk & impact management	• Understand the risk and impact of the transformation and develop formal plans to manage them • Enable stakeholders to SEE and FEEL the compelling reason for the change and reduce resistance to change by minimizing fear, anger, and complacency • Use talent analytics research & tools to assist in finding solutions to the most pressing people-related problems

DevOps Transformation
- Alignment/ Vision
- Pilot
- Adopt
- Scale
  - Train advanced automation topics
  - Support the new Organization Rollout
  - Enable continuous & Quantifiable improvement

Delivering Cultural Change

PDT component		Main objectives
Organization & Talent	**Organization design and governance**	• Realign the organization based on capability, process, and technology changes to optimize resources and employee effectiveness • Align with talent management strategies to determine the right talent, competencies, performance metrics, and incentives are in place to deliver on the business strategy • Develop an operating model that delivers the most value to customers
	Talent requirement and HR programs	• Develop HR strategies, programs, and practices that align with and proactively address the organization's changing talent needs • Create the opportunities, experiences, and guidance that will enable employees to be successfully deployed and connected to their work • Plan for effective reward and recognition of employees for their performance against established individual objectives and contribution to organizational strategy and goals
	Workforce transition	• Understand the new work environment and work processes required as a result of new organization structures and identify the impact on employees • Understand the alignment of roles to the new environment and new work processes in support of responsibility mapping, change impact, and training • Build a trusted network of employees to support engagement across all organizational levels, communicate key messages and obtain feedback
Training	**Learning and knowledge transfer**	• Identify training needs, the amount of training required, and effective training delivery methods that align with infrastructure and culture • Track and validate effective capability transfer through a structured and measured approach based on target capability levels • Develop effective materials that support learning and provide for future reference needs

DevOps Transformation
- Alignment/ Vision
- Pilot
- Adopt
- Scale
  - Train advanced automation topics
  - Support the new Organization Rollout
  - Enable continuous & Quantifiable improvement

Addressing Cultural Aspects

Cultural Aspects

✓ **Address cultural Aspects**

- Inculcate innovative culture to adopt practices around continuous improvement and automation by making sure that the team must focus on :
  - Teaming, collaboration, shared success
  - Automation and continuous improvement
  - Pushing to smaller releases
  - Embracing new technology and frameworks through “System Thinking”

✓ Culture is about embracing a new way of life with common beliefs, practices and values shared by the development community. Deloitte’s **CulturePath*** diagnostic services help define, monitor, sustain and strengthen an organization’s culture during the DevOps transformation. Quantitative dashboards, infographics, and analytics provide real-time feedback and help surface actionable insights for the program management team to address cultural and behavioral changes needed to help ensure sustainable transformation.

✓ In addition, using **ChangeScout*** cloud-based platform, a holistic view of all change-related activities, can be developed including: stakeholder engagement, communications, and training. By monitoring the effectiveness of communication, extensive reporting capabilities and visual dashboards, data-driven decisions can be made to adjust in iterative, agile environments.

Note :- For more details on CulturePath and ChangeScout please visit Organization Transformation Offerings page

Scale: Enable continuous & Quantifiable improvement

DevOps Transformation enables the seamless adoption of DevOps across various levels, i.e., project, product, and organization levels.

DevOps Transformation
- Alignment/ Vision
- Pilot
- Adopt
- Scale
  - Train advanced automation topics
  - Support the new Organization Rollout
  - Enable continuous & Quantifiable improvement

Objective	✓ To establish a metrics-driven mindset that accurately measures and communicates the effectiveness of the DevOps initiative, improving support and buy-in from the business
Detailed Activities	✓ **Define Metrics** ▪ Define a set of metrics which can be used by the business as governance mechanism while promoting healthy competition between teams such as: • Frequency cycle time • Deployment frequency • Change lead time • Mean Time to Recover (MTTR) ✓ **Monitor Metrics and Feedback** ▪ Setup monitoring as it helps identify the problem as it occurs or even proactively identify a potential problem ▪ Setup a continuous feedback loop back from operations to business and development stakeholders

PACE: Moving at the Speed of DevOps

- Accelerate transformation into an Agile and DevOps enabled organization by making **four critical pivots: Process, Architecture, Culture** and **Engineering. PACE**
- **Reduce toil with Site Reliability Engineering**
- Accelerate to Continuous Integration / Continuous Development with Deloitte's plug-and-play **DevOps Cloud Platform**
- Leveraging our **engineering knowledge, industry experience** and **cultural transformation capabilities**

PACE: four pivots on the path to DevOps

Get to DevOps faster, speeding up your development release cycles and improving quality

Developers want **speed**

Operations want **stability**

Everyone wants **security**

Deloitte delivers DevOps with **PACE**

The PROCESS pivot
Ways of working

The ARCHITECTURE pivot
Application architecture

The CULTURE pivot
Organization & workforce

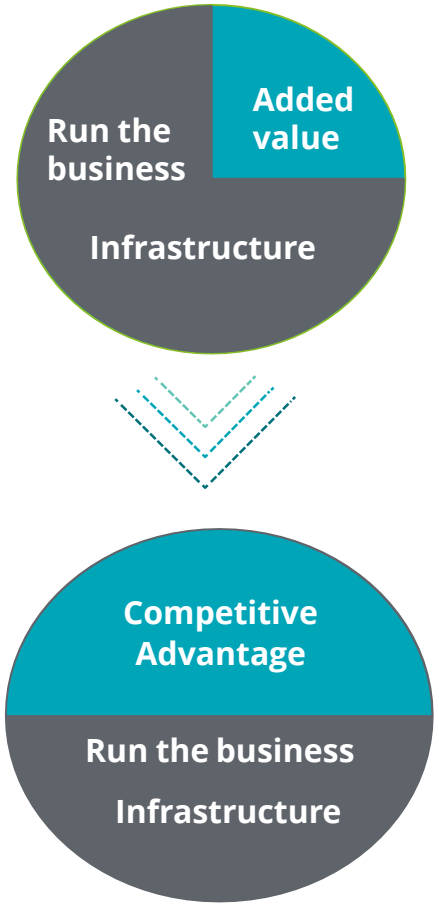
The ENGINEERING pivot
Cloud infrastructure

DevOps delivers speed, stability, and security

Transform your organization into a driver of competitive advantage

The challenge

"Can we **spend less time on toil** and **more time** reducing cycle times and costs, increasing efficiency and responsiveness, ensuring stability and security—and truly creating new **competitive advantage**?"



Today

- Ops and Dev in **silos / separate teams**
- Engineering talent spends **disproportionate time on toil** and not enough time adding value
- **Legacy infrastructure** and application management **devours investment resources**
- Team and Technology **change resistant**, slow to provision and **hard to maintain and scale**

Tomorrow

- **Talent integrated** in smaller, autonomous teams blending software engineering and operational skills with **shared accountability** for development and **stable, secure operations**
- **SRE** (Site Reliability Engineering) offers pragmatic new ways of working, enabling dramatic **improvements in release cycles**
- **Automated** application migration to the cloud
- Replatformed and refactored applications deliver **nimble, scalable**, cloud-native microservices at **enterprise scale**
- Operate using an **automated, scalable, agnostic infrastructure**—managed as code, offering lower total cost of ownership
- Improved **time to value**

The PROCESS pivot

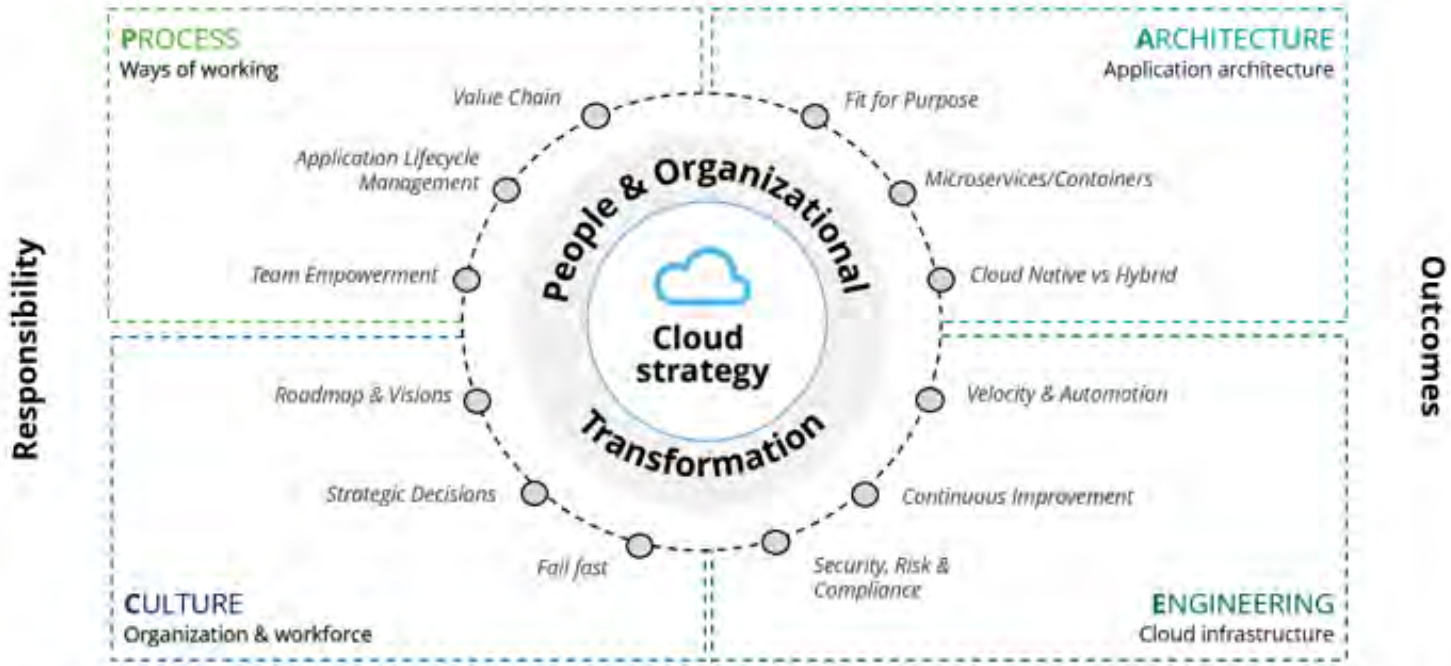
Ways of working

Agile, iterative, measured, improving

The ARCHITECTURE pivot

Application architecture

Cloud (single, multi or hybrid) microservices, serverless, containers



The CULTURE pivot

Organization and workforce

Empowered teams

The ENGINEERING pivot

Cloud infrastructure

Infrastructure automation, CI/CD

The PROCESS pivot

Ways of working

Integrated, Agile engineering + operations

Accelerated, automated and integrated delivery

Today – Legacy delivery and operational processes require high manual labor (toil) and administrative overhead. Developers and operations teams operate separately

Tomorrow –

- Redesigned processes enable sprint featured release cycle times as the new norm
- Agile iterative delivery through sprints
- Small batches and frequent releases
- Minimum Viable Product (MVP) approach, limit work in progress for increased flow

The ARCHITECTURE pivot

Application Architecture

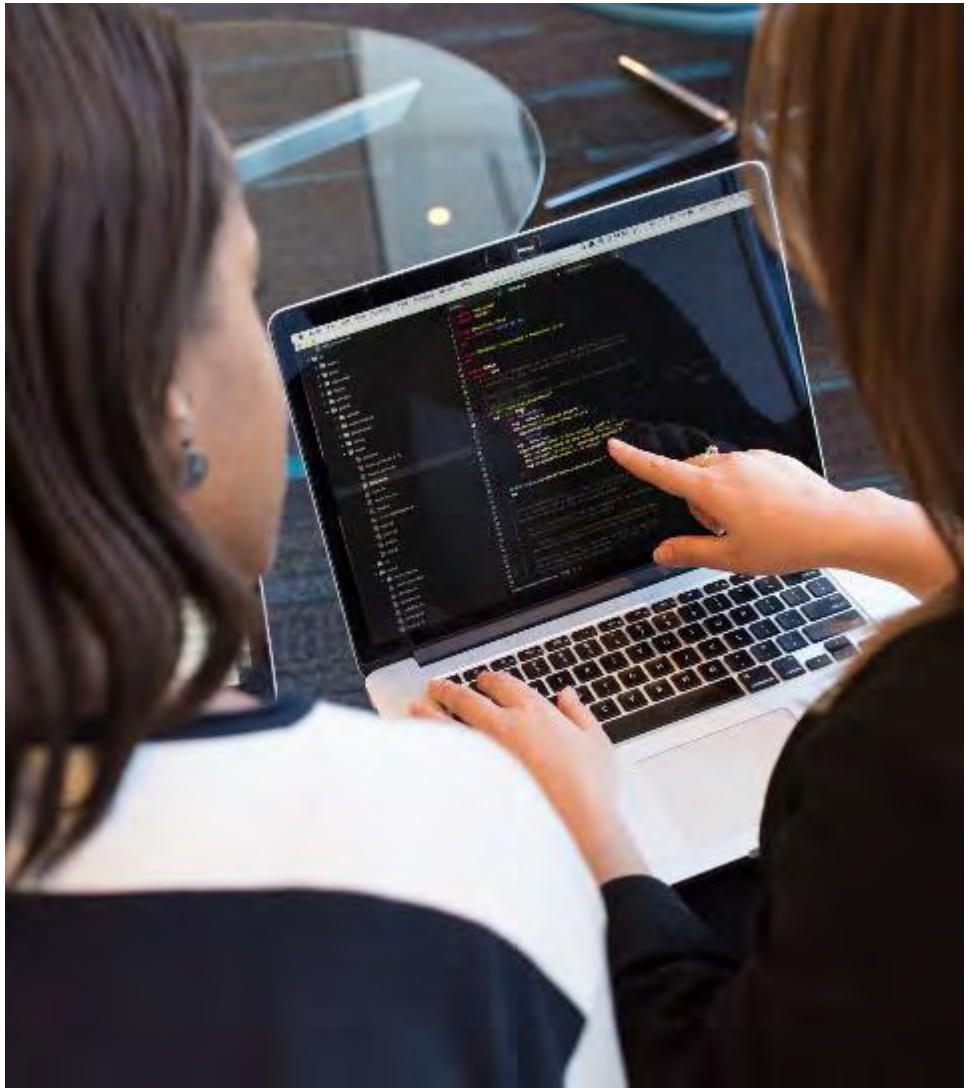
Cloud (single, multi or hybrid)
Microservices, Serverless, Containerization

Migration of replatformed / refactored apps to cloud

Today – Technical debt across thousands of legacy applications eats up large portion of spend and resources for operations, maintenance and infrastructure

Tomorrow –

- Automated application migration to the cloud with reduced manual process
- Mission-critical applications are architected or replatformed/refactored into nimble, secure, scalable microservices
- Integrated teams spend more time coding and innovating, less time overseeing existing infrastructure



The CULTURE pivot

Organization & workforce

Empowered Teams

Move toward smaller, autonomous engineering teams with new skills

Today – Legacy workforce lacks the capacity, skills and organizational structure to undertake modernization

Tomorrow –

- higher ratio of engineering talent working in small, customer-focused teams
- Shift left where greater responsibility is put on the autonomous teams to make decisions
- Accept failure as part of the process
- Breakdown organizational silos between development team and operational teams
- Teams work directly with the product owners and have greater visibility into the business

The ENGINEERING pivot

Cloud infrastructure

Infrastructure automation, CI/CD

Managed as code in your cloud

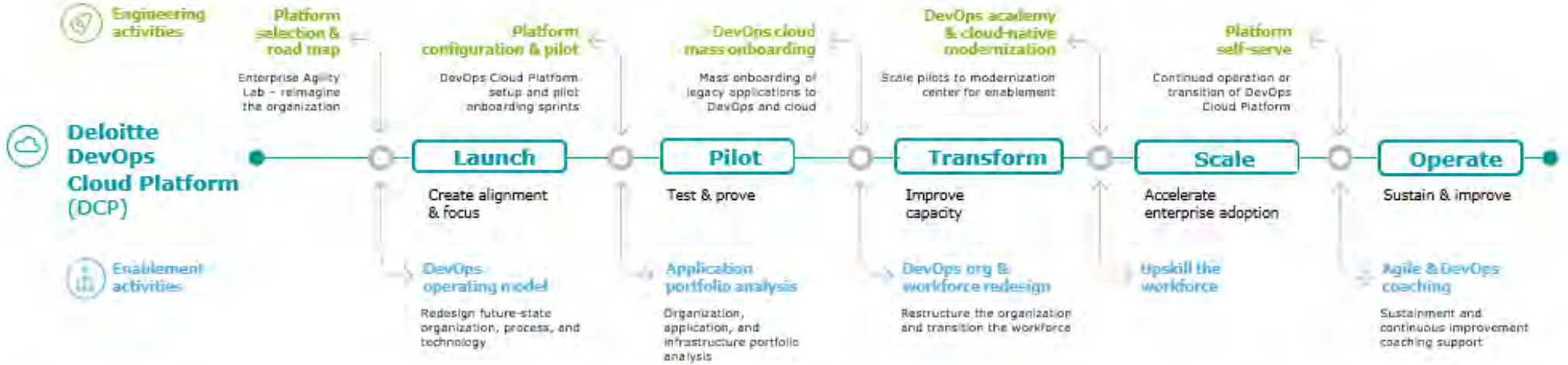
Today – High manual labor costs and high costs of legacy infrastructure are change resistant, slow to provision and hard to maintain and scale

Tomorrow –

- Implement application containerization on existing and new application architecture
- Dynamic and nimble application architecture to perform self monitoring, compliance and provisioning of infrastructure
- Physical and virtual silos between security, QA+QC and automation are eliminated, united by and managed as code
- Deploy small batches of changes to accept failure and learning as part of the process

The PACE of DevOps transformation is driven by a powerful platform

Engineering and enablement transformation efforts are both accelerated using **Deloitte Cloud Platform**



At each step of the transformation journey, we embed considerations across the four dimensions of PACE:

Process

Ways of working
Agile, Iterative, Measured, Improving

Architecture

Application Architecture
Cloud (single, multi or hybrid) Microservices, Serverless, Containerization

Culture

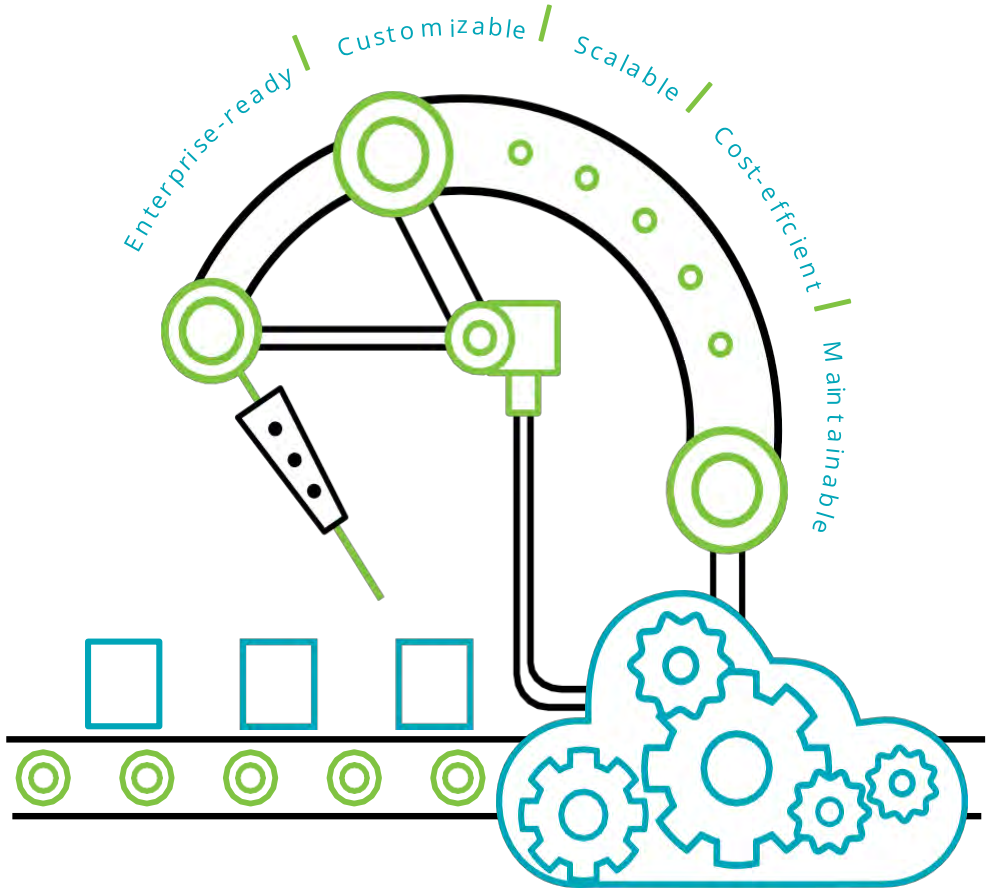
Organization & workforce
Empowered Teams

Engineering

Cloud infrastructure
Infrastructure automation, CI/CD

DevOps Cloud Platform

DCP is a powerful accelerator on the path to speed, stability & security—integrating industry + culture + engineering



DevOps Cloud Platform (DCP)

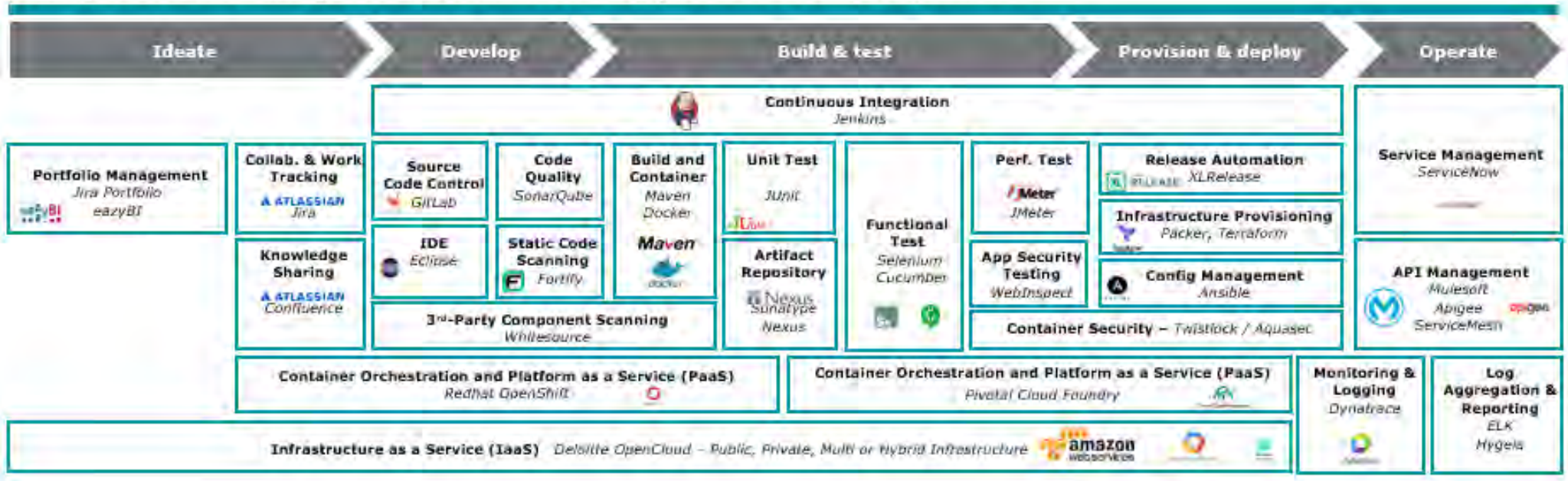
Integrated and configurable DevOps Cloud Platform with built-in industrialized full-service CI/CD and security capabilities that is application ready

DCP is available in all environments and can be customized to fit your infrastructure design

DevOps Cloud Platform

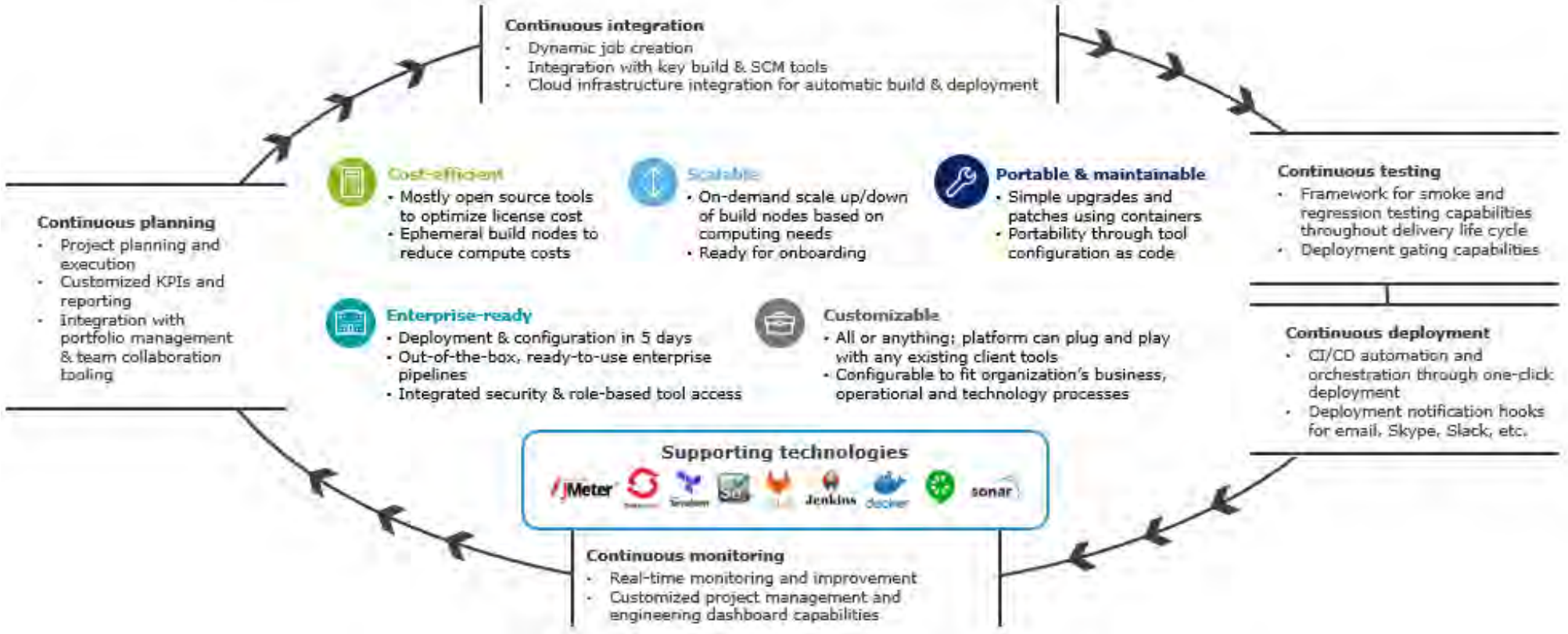
Integrated DevOps platform using cutting-edge DevOps tools codified with our leading DevOps practices into the configuration

Configurable and Integrated Toolchain with Application Ready Capabilities

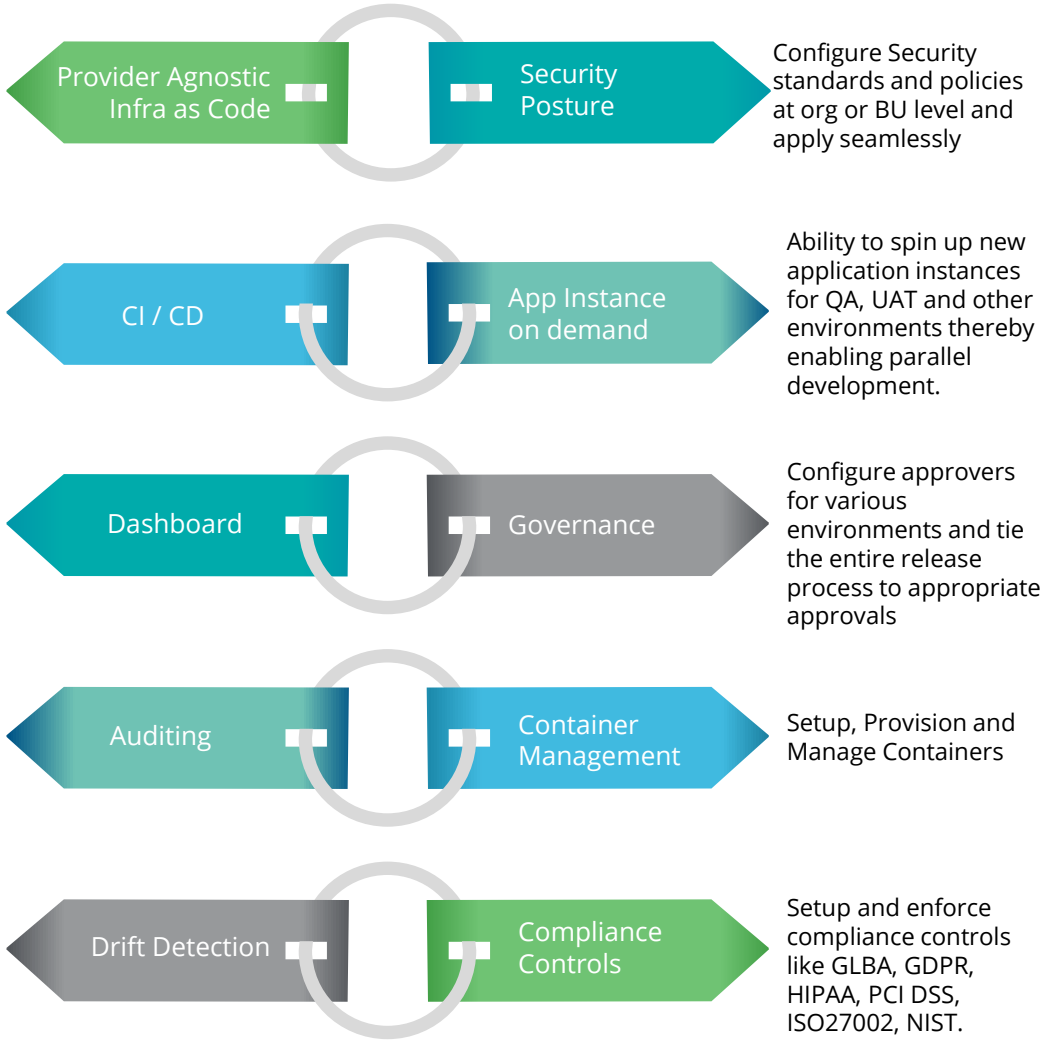


Achieving PACE: a continuous environment

Integrated DevOps teams use DCP to plan, integrate, monitor, deploy, and test to ensure stability, security, and speed



And adds High Value Functions through the Platform



DevOps Cloud Practice key contacts

Bob Vuong
Managing Director
bobvuong@deloitte.com
+1 703 829 0697

Jay McDonald
Managing Director
jaymcdonald@deloitte.com
+1 781 980 0232

Eddy Krumholz
Specialist Leader
ekrumholz@deloitte.com
+1 303 305 4881

Jane Cavanaugh Snider
Senior Manager
jsnider@deloitte.com
+1 612 834 8688

Tony Witherspoon
Specialist Leader
twitherspoon@deloitte.com
+1 571 766 7029

Viz

Agile	Source Code & Version Control	Build Automation	Continuous Integration	Test Automation	Code Quality	Release Automation	Continuous Monitoring
• Confluence • Jira	• Git • Bitbucket	• Maven • MS Build • Artifactory	• Bamboo/Jenkins	• Junit • Selenium • LoadRunner • Jmeter • VS Test	• SonarQube • Veracode	• Udeploy • Docker • MS Deploy • Artifactory	• Lighthouse

ETL

Agile	Source Code & Version Control	Build Automation	Continuous Integration	Test Automation	Code Quality	Release Automation	Continuous Monitoring
• Confluence • Jira	• Git • Bitbucket	• N/A	• Bamboo/Jenkins	• N/A	• FCAT	• FCAT	• Lighthouse

Big Data

Agile	Source Code & Version Control	Build Automation	Continuous Integration	Test Automation	Code Quality	Release Automation	Continuous Monitoring
• Confluence • Jira	• Git • Bitbucket	• Maven • Artifactory	• Bamboo/Jenkins	• Big Data Test Engine • ScalaTest - Spark Test Base	• Scalastyle • SonarQube	• Bamboo/Jenkins • Artifactory	• Lighthouse

When a global pharmaceutical company needed assistance in improving their product delivery capability Deloitte was there to help.

DevOps Strategy

Coaching

Automation

Issue

The client is a global pharmaceutical company that wanted to improve the product quality and reduce the time to value. Deloitte did a current state assessment and identified the following issues: difficulties in communication and collaboration between internal teams, challenges in onboarding multiple stakeholders for multiple modules, frequent scope changes during development resulting in delayed delivery timelines, testing new builds took a long time for completion and client commitment to the development process was not at desired levels

Solution

Deloitte recommended a transformation strategy for the client to remain competitive in the Health Care Market. Deloitte provided support in the following areas:

- Methodology and Process Automation - Defined and helped setup end to end DevOps model for application development and deployment. Tools were used to automate testing, project estimation and quality management reducing release time
- Tool Assessment - Validated and shortlisted tools for adoption – build, deploy, test & report.
- Helped with execution of SCRUMs and enabled processes for increased collaboration across teams
- Training & Coaching - Demonstrated to practitioners on the importance of collaboration between development and operations and coached them on adopting it
- Continuous Monitoring - Helped setup centralized monitoring dashboards to check for real time updates

Impact

- Reduced release time using tools to automate testing, project estimation and quality management. Sprint timelines of 2 weeks helped reduce the overall turnaround time. Proactive quality management helped in on-time delivery of solutions at lower costs.
- Daily interaction with business helped align development sprints to user requirements.
- People embraced a culture of innovation and shared learning due to the extensive onboarding sessions that were held to set expectations and identify processes
- DevOps managers had centralized monitoring dashboards to check for real time updates
- Enhanced collaboration between teams lead to greater transparency. E.g. Support and release teams were kept informed of upcoming projects.

When a high tech organization needed assistance defining their Continuous Integration and Continuous Delivery strategy, Deloitte was there to help.

DevOps Strategy

Automation

Issue

The client is a high tech organization and as part of its transformation strategy assessment, identified the following issues: 60% of non-production issues and 40% of production issues were related to configuration changes, no automated process existed for continuous build, deployment and testing, multiple manual touch points in build, added to the overall 'time to build', dependencies were not managed, leading to lower build success rate, no visibility and integration during the build, deploy & test increases.

Solution

Deloitte recommended updates to the client's configuration management system and code deployment processes and helped enable the following:

- Configuration as Code
  - Consolidated multiple deployment strategies followed across verticals
  - Built a centralized configuration management framework to eliminate manual touchpoints to add / edit configuration files
  - Leveraged light weight containerized approach to enable "Build once…run anywhere and Configure once…run anything"
- No Touch Deployment
  - Conducted fitment analysis through narrowed-down Proof of Concepts using Continuous Integration / Continuous Delivery tools
  - Helped set up automated build pipelines with static code analysis, code coverage, test automation & sanity packs
- Built centralized monitoring for increased visibility into the release process
- Helped adopt a test driven development methodology

Impact

- Improved Agility, reduced overall number of Software Development Life Cycles and provided faster turn around time
- Cost savings, reduced delivery cycle through process automation and improved quality by eliminating manual touchpoints
- Increased visibility, centralized monitoring provided greater visibility to track Continuous Integration Build, Quality Engineering process, and helped with quicker reaction time to issues
- Improved accountability, implemented process to collect actionable metrics for insights into development and code quality for faster development

Bringing Agile and DevOps Together

Agile and DevOps are two sides of the same coin that share common principles and focus on breaking the wall between Business and IT and the wall between Dev and Ops.

Agile & DevOps PRINCIPLES

- **Collaboration:** Traditionally siloed teams serve "one-common" purpose
- **Customer Satisfaction:** Achieved by reduced time to market
- **Frequent Releases:** Working software is released frequently into production
- **Automation:** Code and Infrastructure are automated and testing shifts left
- **Communication:** Face-to-face conversation is considered most efficient and effective
- **Transparency:** Quality is built-in with full transparency at every stage
- **Working Software:** The primary measure of progress is the working software

Agile PRACTICES

- Cross-functional Team
- Continuous Funding
- Customer Centric Delivery Model
- Test & learn mindset
- MVP – Value-based Development
- Autonomous Decision Making

DevOps PRACTICES

- Continuous Integration
- Application Portability
- Automated Testing
- Continuous Deployment
- Dynamic Environments
- Continuous Monitoring

AGILE DEVELOPMENT

Moving from business ideas to the end of software development

DEVOPS DEVELOPMENT-RELEASE PROCESS

From value creation through delivery to the end user

SRE – In a Nutshell

SRE ensures the stability of the production environment and at the same time is committed to push for rapid changes, new features and operational improvement

Site Reliability Engineering

- SRE is a single **point of responsibility and arbiter between the Dev and Ops teams** to ensure reliable and low latency applications in a continuous delivery environment
- SRE also focus on **building and maintaining automation pipelines**, to do away with repetitive tasks as well as production support to applications, reducing the time and level of effort needed for ongoing support
- SRE typically **combine with software development, networking and system engineering expertise** to build and run large scale, massively distributed, fault tolerant software systems and infrastructure.

Roles and Responsibilities

SRE team is responsible for **availability, latency, performance, efficiency, change management, monitoring, emergency response, and capacity planning**

Typically, SRE spend their time doing:

Operations (~50%)

- Proactively monitor systems
- Work support issues
- Develop playbooks and best practices
- Create items for the development backlog

Development (~50%)

- Fix production issues
- Automate manual tasks
- Experiment new methods for improving resiliency
- Collaborate with development team

*** Note :- For more details SRE topic please refer SRE Point of View**

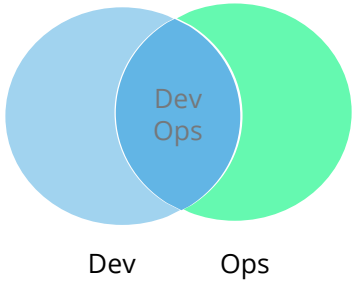
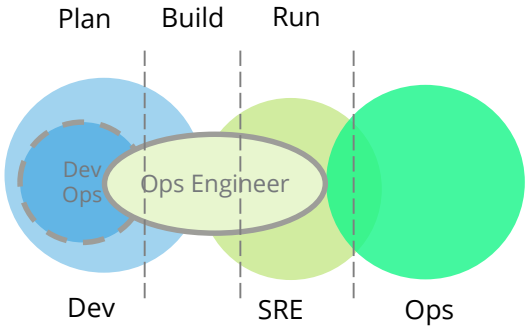
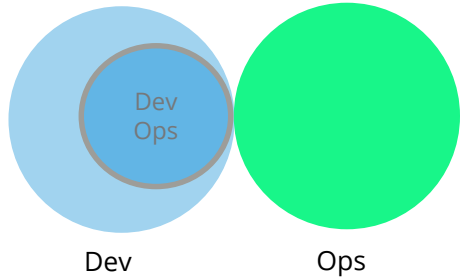
How is SRE linked with DevOps?

SRE is a “amplifier” for DevOps teams to move faster and enables increased productivity and efficiency

Pillars of DevOps	What does it mean for DevOps	How does it translate to SRE	What does it mean for SRE
Reduce Organizational Silos	Break down barriers across teams and increase collaboration	**Shared Ownership**	Everyone working with the app has same view by using same tools
Accept Failure as Normal	Computers and Humans together in a system is likely to introduce imperfections	**SLOs and blameless PMs**	Same failure is not repeated and the error budget decides how much system can go out of specs
Implement Gradual Change	Gradual changes are easier to review and roll back if those don’t go well	**Reduce cost of failure**	Roll out apps to small percentage of users before moving it out for all users
Leverage Tooling and Automation	The more the automation, the less is the manual effort and errors	**Automate the job**	Trying to eliminate the manual efforts as much as possible
Measure Everything	The only way of knowing success is by measuring it	**Measure efforts to keep system reliable**	How much efforts go in for maintaining the reliable health of the system

If DevOps is a philosophy, SRE is a prescribed way of accomplishing that Philosophy

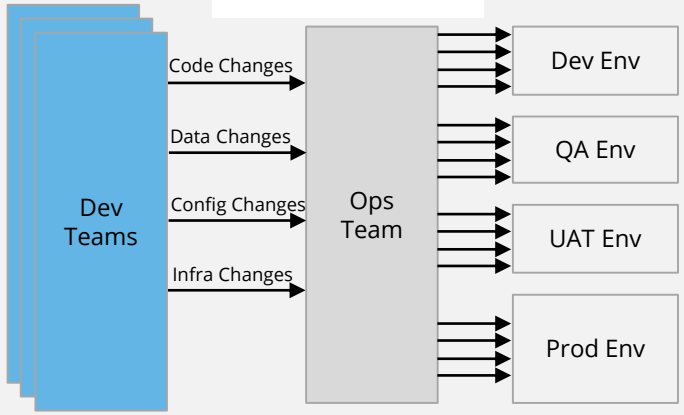
Typical Patterns DevOps-SRE Adoption

	Dev & Ops Collaboration (Ideal Long Term Solution)	SRE Model (Easier to Implement)	Ops As IaaS (Preferred Option)
Model			
Description	• DevOps is a collaboration between Dev and Ops, working for and belonging to both functions • This is true build-it/run-it with Dev and Ops teams working toward the same application goal • Ops team is a combination of resources skilled in ops, engineering, architecture and security • In some cases, quality assurance and security teams may also become more tightly integrated with development and operations throughout the application lifecycle	• A maturity of previous pattern, instantiating a Site Reliability Engineer (SRE) between Dev and Ops teams • Consists of clearly defined Plan, Build and Run phases • The Dev (and DevOps) teams contribute to the highest degree in the Plan phase, with the Ops Engineer focusing on the deploying a stable build • The SRE runs the approved project build on field to identify and address issues promptly • The SRE role is a deviation from pure play DevOps and has experience in cloud infrastructure and cloud applications	• Offers a traditional shared infrastructure services feel to Dev teams • A Small DevOps team sits within the Dev team and provide liaison support to Dev teams for IaaS related features provided by Ops (e.g. monitoring, provisioning etc.), thereby focusing on full-fledged development & operations • The ops team does not follow a full agile based methodology in their approach. They focus solely on op based activities

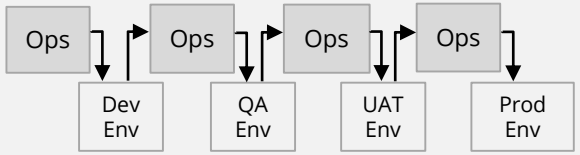
Riding the DevOps Maturity Curve

For successful DevOps adoption organizations should shift towards cross-collaborative, highly dynamic teams that can execute and maintain automated processes

Traditional IT



Workflow to Production



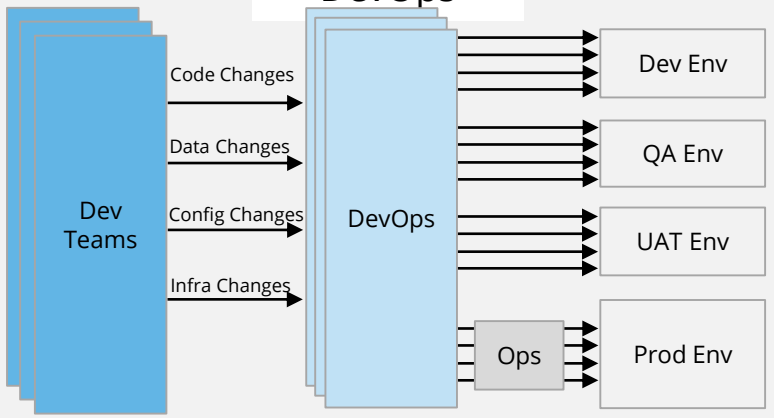
None

Developer Autonomy

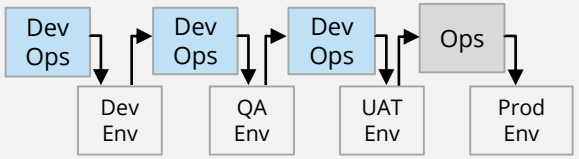
	Design	Ops
Apps	Dev Team	Managed AppOps
Infra	Internal Ops Team	Managed InfraOps

Accountability Matrix

DevOps



Workflow to Production



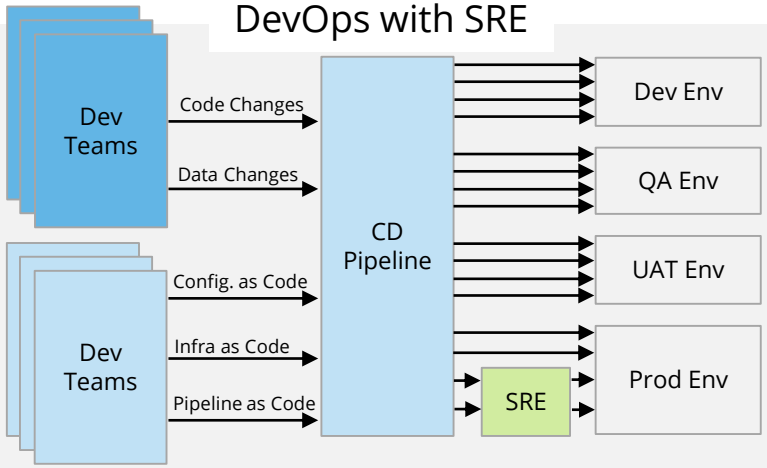
Low

Developer Autonomy

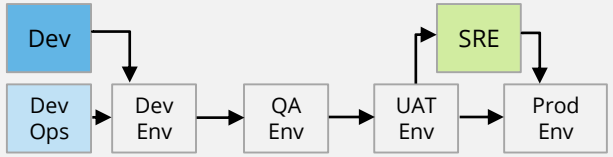
	Design	Ops
Apps	Dev Team	DevOps
Infra	Internal Ops Team	

Accountability Matrix

DevOps with SRE



Workflow to Production



Medium

Developer Autonomy

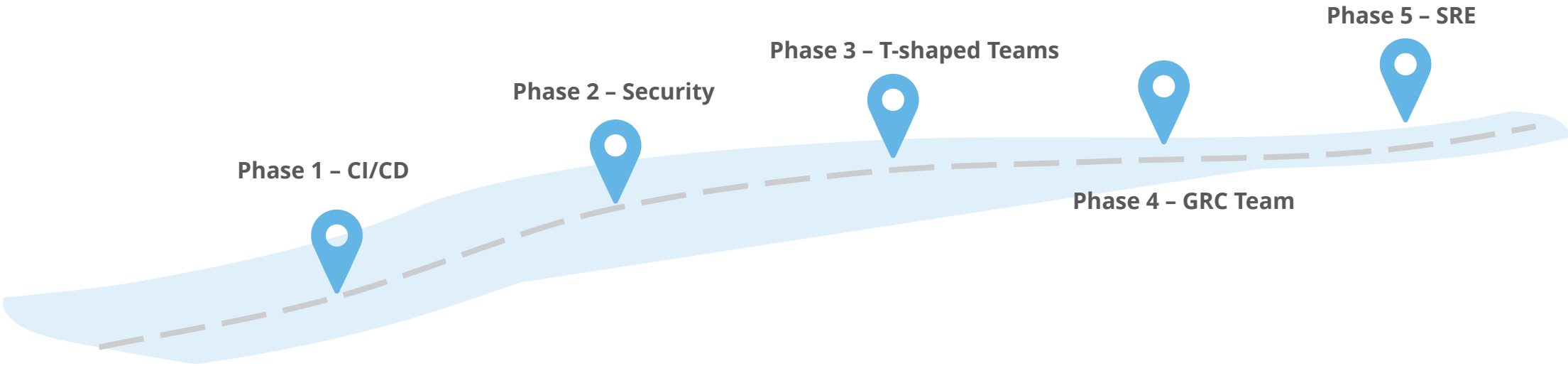
	Design	Ops
Apps	Dev Team	
Infra	SRE Team	DevOps

Accountability Matrix

A full scale DevOps adoption would require incubation of SRE (Site Reliability & Engineering) which focusses on developer empowerment and operational improvement while ensuring stability in the production environment

DevOps Maturity Curve Evolving to SRE

SRE is all about achieving operational excellence and requires a combination of depth and breadth of organizational maturity



Phases	CI/CD	Security Automation	T-Shaped Teams	GRC Team	SRE
Bottleneck	• Non-repeatable error-prone build process • Provisioning times and inconsistent bottlenecks	• Security	• Bottlenecks due to manual efforts	• Governance, Risk and Compliance	• Support structure from Tier 1 to Tier 3
Solution	• *Continuous Integration* • *Continuous Delivery*	• *Upfront built-in Security* • *Security Automation*	• *T-shaped Teams* • *Run what you build*	• *Separate policy ownership from policy implementation*	• *Move the problem closest to the people who can solve it*

Containers – Overview

Containers are a method of operating system virtualization that provides the necessary computing resources to run an application independently on a shared OS

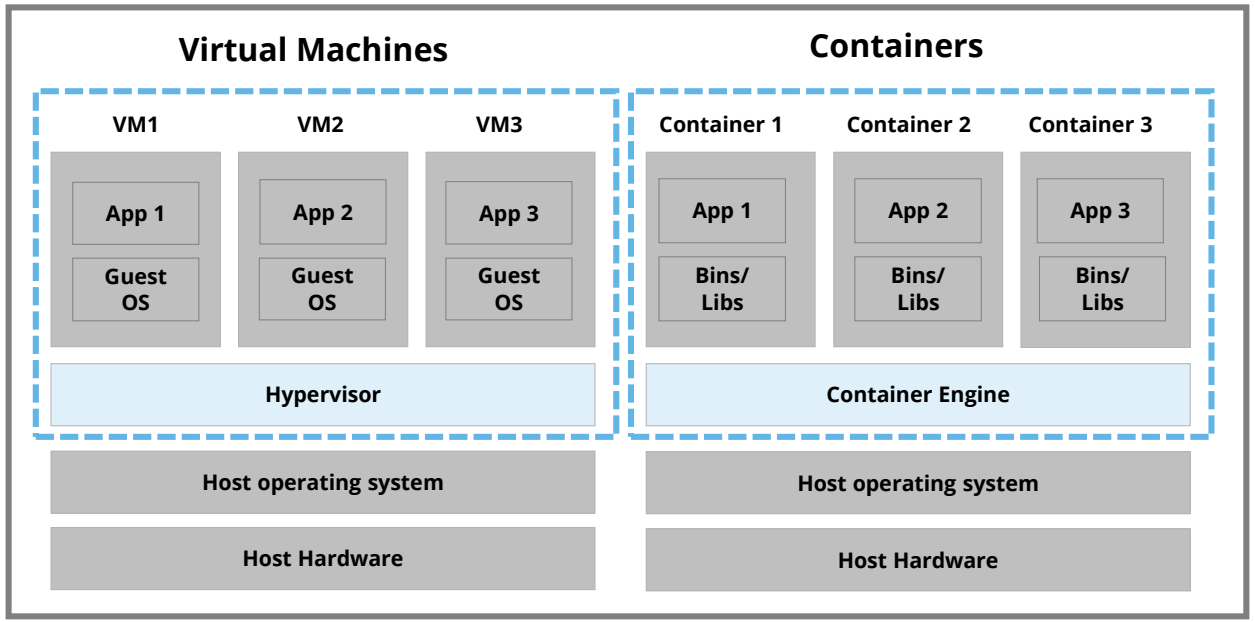
Key Characteristics

- Intended to run a single application
- Bundles all application dependencies such as libraries, binary files, and other configuration files into a package, making it portable
- Creates isolation boundary at the application level, allowing multiple applications to reside and share the same OS
- Created containers can be deployed to different servers, leading significant software lifecycle benefits

Difference between Virtual Machines and Containers

- The key difference between containers and virtual machines is the way the operating system resources are used and the location of the virtualization layer
- Each virtual machine runs a unique OS , whereas containers shares the host OS
- Virtual machines might take several minutes to boot up, but Containers take only few seconds

	Virtual Machine	Containers
Start Time	30-45s	<50ms
Stop Time	5-10s	<50ms
Workload Density	1x	10-100x



*** Note :- For more details containers & microservices topic please refer Microservices and Container Management POV**

Containers – Architecture

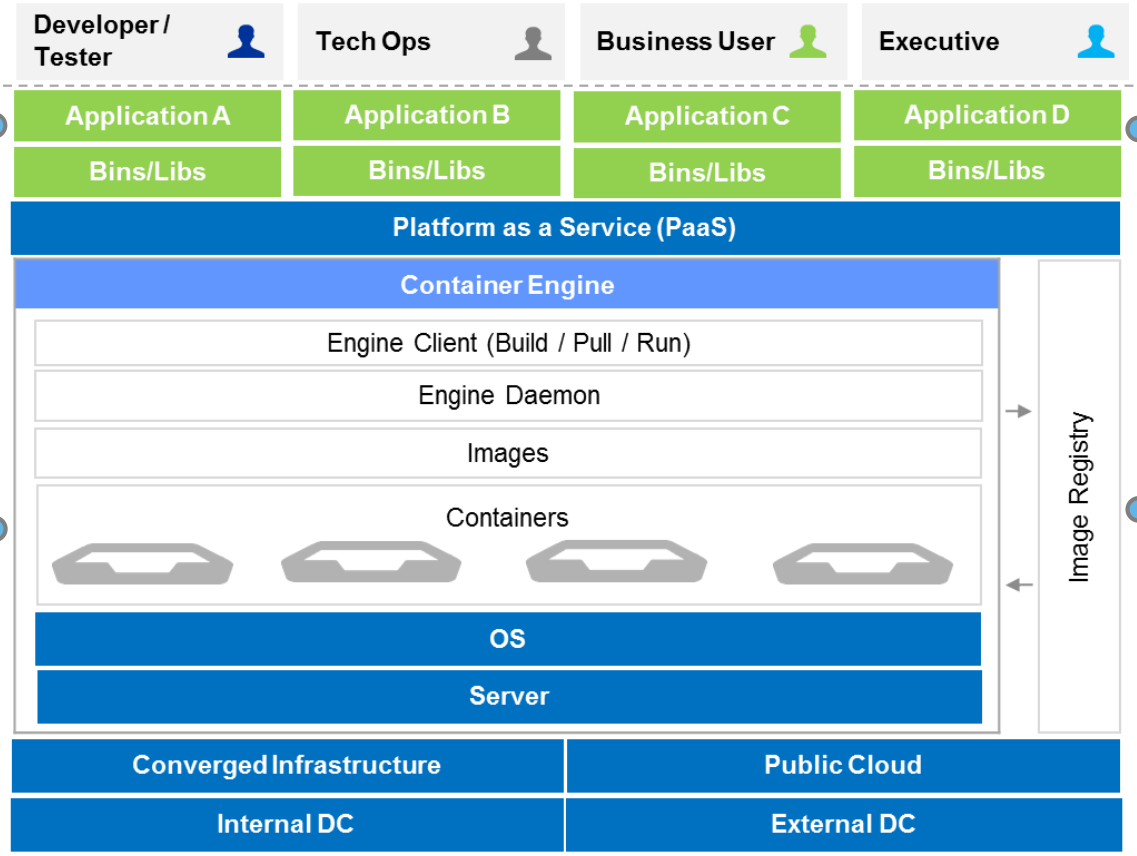
Containers provide a lightweight virtual environment that groups and isolates a set of processes and resources such as memory, CPU, disk, etc., from the host and any other containers.

App Portability Service:

Frees developers from software and infrastructure dependencies, cutting costs and creating efficiencies in the process

Faster Boot Service:

VMs must retrieve 10-20 GBs of an operating system from storage. The workload in the container uses the host server's operating system kernel, enabling it to boot faster



Process Isolation Service:

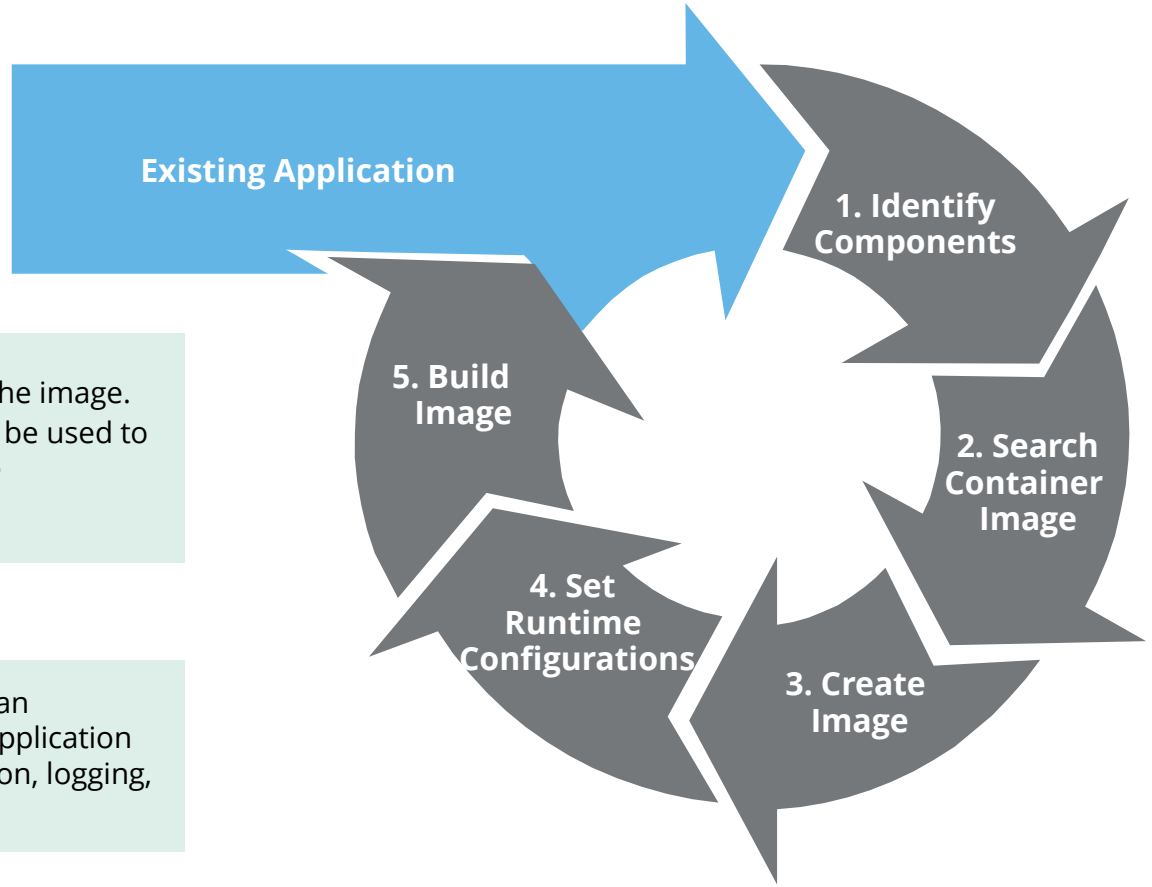
Runs as an isolated process in the user space on the host operating system, sharing the kernel with other containers

Lighter Weight Infrastructure Service:

Containers share the host's OS and are therefore lighter in weight.

Containerization of Applications

Containerizing an existing application involves certain key implementation steps that are generally common and necessary, irrespective of the platform and vendors



Identify the different components of the application such as framework, external services such as database, proxies, etc. that it relies on. These services could be internal or external

Searching for potential images for each components to base an application from repositories such as Dockerhub, Amazon, GitHub, etc.

Create container image. Two options:
1. Bundle all the components into one image
2. Create different images for different components.

Set up runtime configuration for an application which could include application settings, security, database location, logging, etc.

Assemble all the code and build the image. In case of Docker, Docker file can be used to build an image and publish in the DockerHub.

- *Best Practices #1: Create different container image for application components whenever possible*
- *Best Practices #2: Application and database should be in different containers and not packaged into one container*

Containers – Key Considerations

Adopting container technology would require organizations to conduct due diligence with regards to existing infrastructure, security and cloud technologies

Hosting Platforms	Identifying **bare-metal servers or virtual machines as hosting platforms** for containers. Bare-metal provides higher performance, easier administration, scalability, etc. On the other hand, VM provides isolation, consistent software environment, etc.
Hosting Environment Setup	Maintaining the **infrastructure configurations standardized** across development, production and testing environments for consistent and smooth performance of containers which includes choice of kernel, user roles, network and firewall, etc.
Security	Sharing OS, hardware and other computing resources and also contents within the container application raises security concerns. Mitigate security concerns by **e**nsuring the source of the contents inside containers, conducting regular inspection of container contents and using components from trusted sources.
Storage	Storage can be shared between containers or isolated to each container. **Persistent storage is critical** to ensure that the data is safe and secure even after the container has stopped working.
Logging & Monitoring	**Dynamic setup is preferable** over static to monitor container activities. The monitoring and logging system should ensure that the log data is collected and stored safely.
Cloud Delivery Model	Container technology might lead organizations to change to multi-cloud or different cloud solution. The flexibility of the cloud service currently being used within the organization would help in the smooth adoption of containers.
Native Solutions	Implementing **container native solutions provides many benefits** and eases the use of containers in the production environment. It treats containers as first class unit of infrastructure and not the virtual machine or physical machine.
Container Management	**Management of containers internally or through third party vendors is pivotal**. Some vendors handle management on it's own and provides a broad range of features.

Microservices – Overview

Microservice architecture is an approach of developing software applications as a suite of small modular services, each running its own unique process to support a specific business function

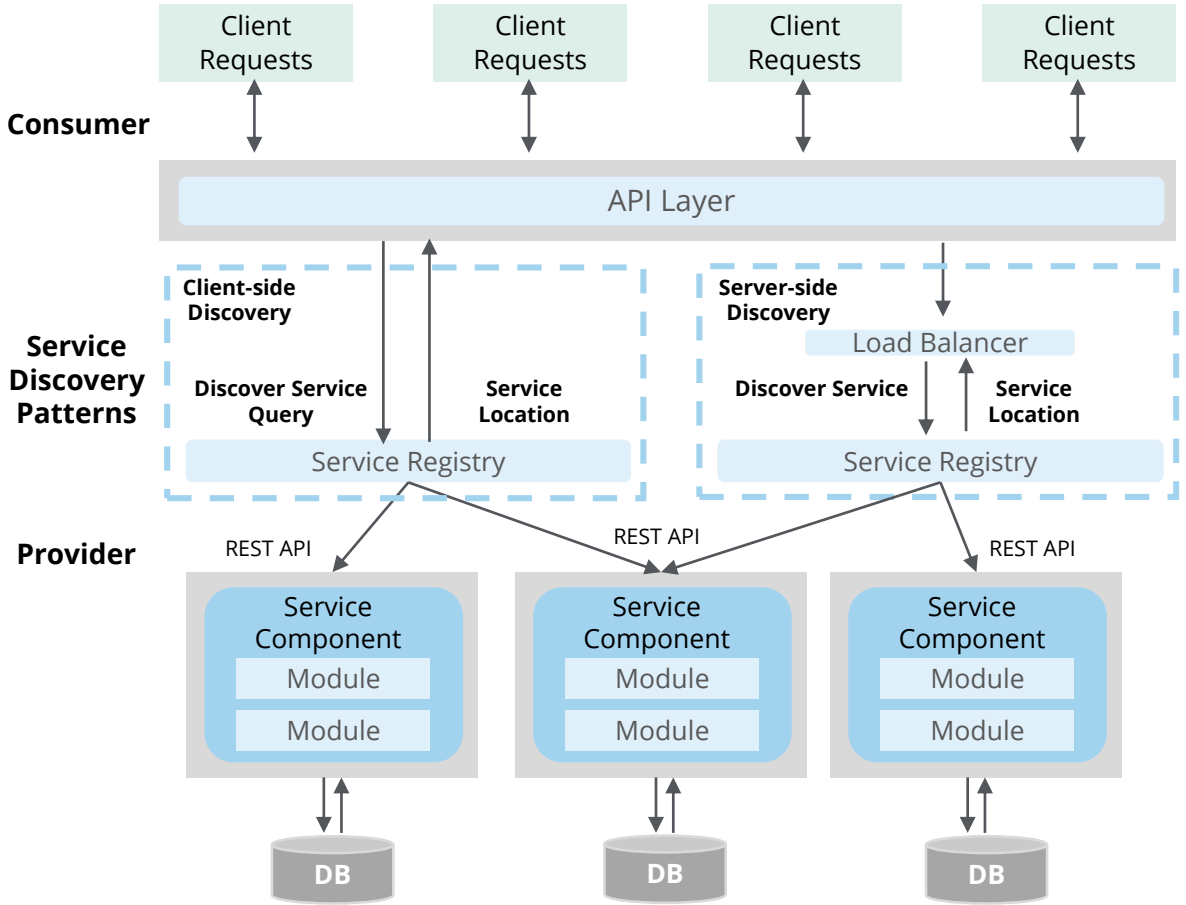
Microservice Application

- Divided into several small services
- Each service responsible for running a specific functionality
- Each service is independent of the other
- Each service can be deployed individually and on smaller hosts containing only required resources
- Each service can be scaled individually based on the resource requirements
- Each services can be developed using different languages and frameworks

Key Challenges

- Increased operational and deployment complexity
- Security of deployed services
- Monitoring, tracking and testing of services
- Handling service dependencies and inter-communication

Microservice Architecture



*** Note :- For more details containers & microservices topic please refer [Microservices and Container Management POV]**

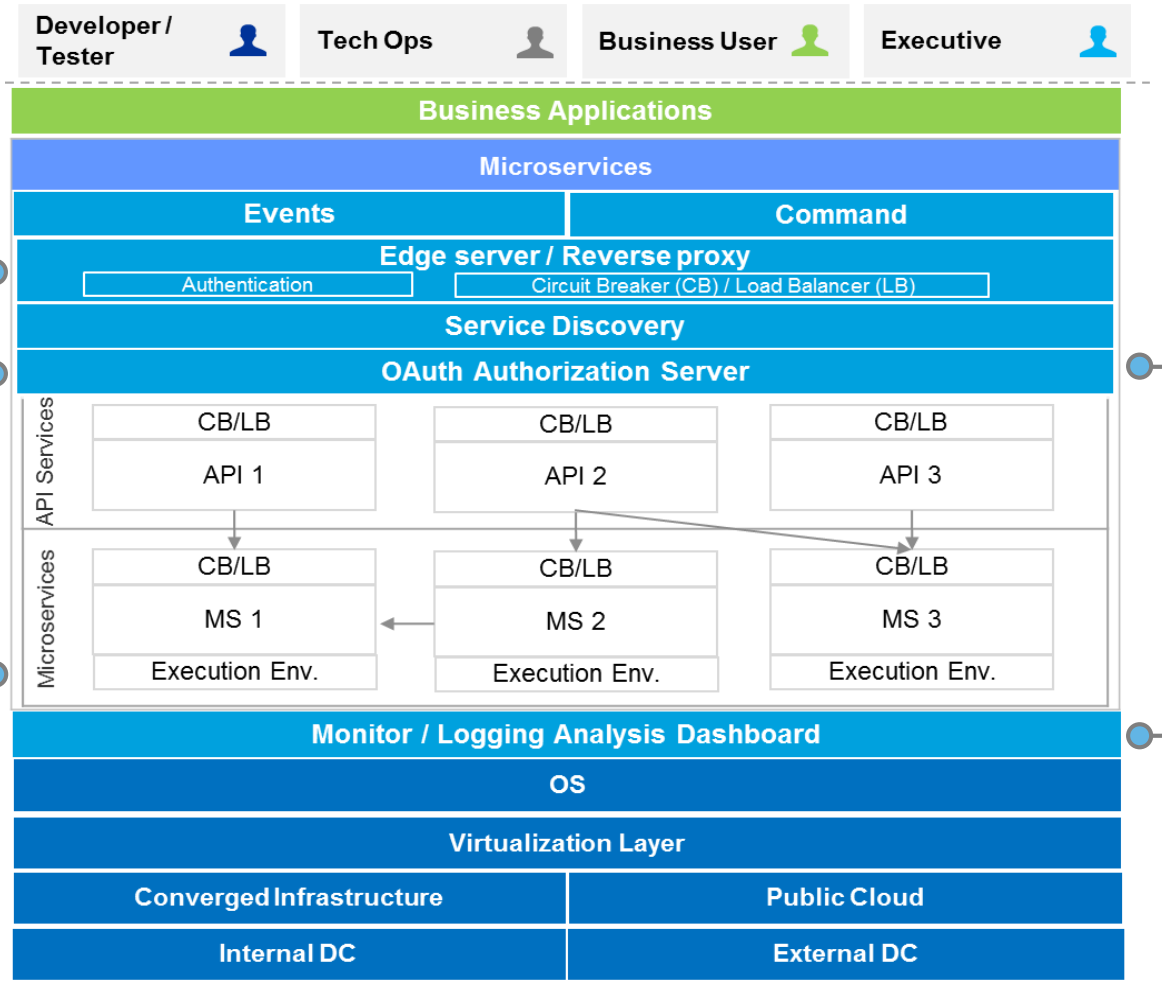
Microservices – Architecture

A microservice is a highly scoped, loosely coupled, strongly encapsulated, independently deployable and scalable application component that enables agility and scalability. Key benefits include faster time to market, reusability, scalability and reduction in break / fix.

Edge Services:
Acts as a dynamic proxy for all the external traffic (prevents unauthorized access)

Security Services:
Acts to protect exposed API services

Composite / Core Services:
Handling persistence of business data and applying business rules and other logic to perform a common task

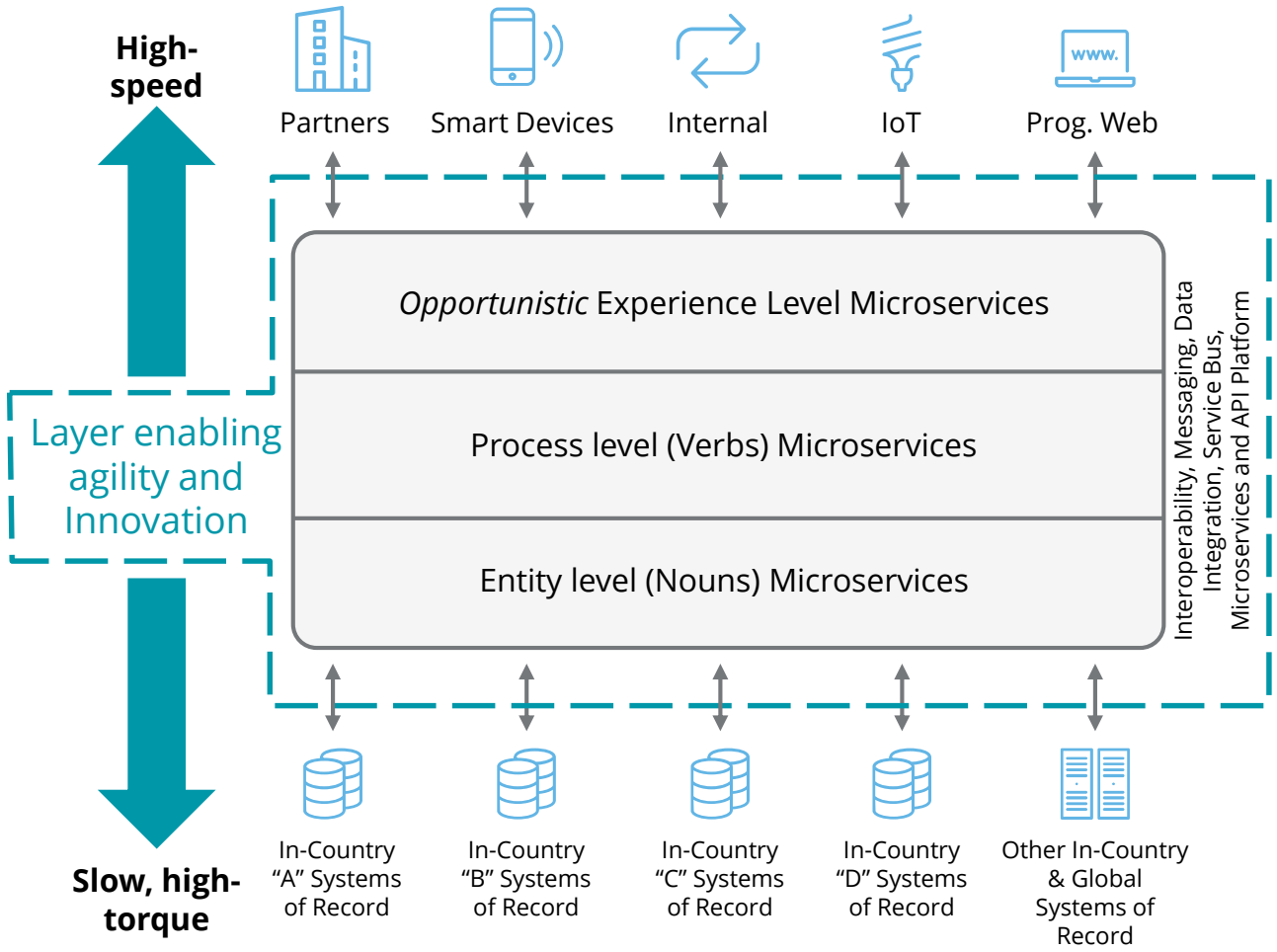


Registration Services:
Allows MS to self register at startup, reducing manual efforts

Health and Usage Monitoring Service:
Centralized log analysis of each Microservice

Microservices – Advantages

Microservices provide a seamless experience across all channels, and the ability to create new customer experience and offering without touching the 'core' every time.



Create a rich, integrated customer experience framework:

- Enabled by an opportunistic API experience layer
- Deliver a relevant, seamless customer experience with channel handoff capability
- Move towards a single responsive design (in addition to native OS designs)

Ability to quickly respond to business demands by:

- Creating interoperability layers promoting innovation and agility with governance & security
- Organizing API's by logical level's to enable standardization, re-use, and multiple-speeds of delivery
- A microservices-based architecture, focused on connectivity, orchestration and transformation
- De-coupling tight dependencies between legacy systems, services and consumers

Microservices – Characteristics and Considerations

Below are key characteristics of microservices. Also listed are preliminary questions which need to be addressed in near term prior to microservice adoption.

Microservices	
Key Characteristics	▪ In a microservices architecture, an application is comprised of a number of small, independent services that interact through an external published protocol, such as REST, or a messaging service ▪ Microservices allow for small, independent, targeted teams that focus on a specific microservice instead of a large, monolithic, complex team devoted to a large, monolithic, complex application ▪ PaaS is crucial for microservices implementation as it not only standardizes environments and significantly reduces complexity, but it also provides a foundation for the major organizational changes required to move to a microservices approach

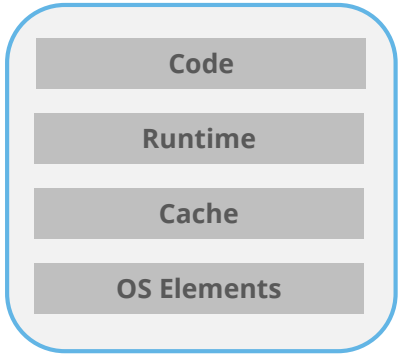
Key Considerations / Questions to be Addressed	
1	What are the key drivers to use microservices within the organization?
2	Are we mature enough to adopt microservices? If yes, what is the level of app modernization and recoding required?
3	Have we selected a microservice vendor/tools? What is the rationale behind selection?
4	How will monitoring/logging services change due to microservices?
5	Has the organization identified an internal microservice owner? Do we need to establish a microservice working group?
6	Do we have any metrics to evaluate the value add of microservices to the end state?

Containers and microservices work in concert with each other

Containers enable microservices and provides an ideal environment to deploy services, leading to scalability, isolation, portability and faster time to package and deploy

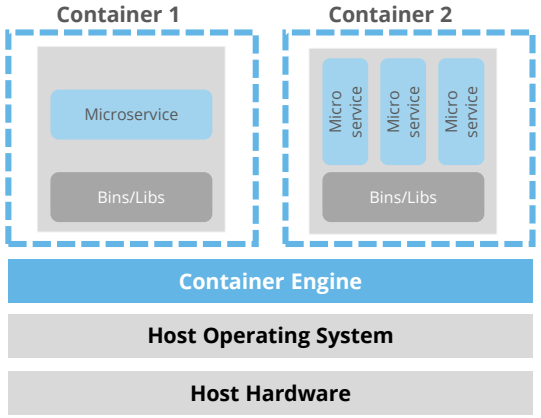
Microservices in Containers

- Elements of microservices environment includes runtime, cache, operating system, etc.
- When implemented with containers, this environment is provided by them.
- Containers keeps tracks of the various activities of microservices with the help of the available tools



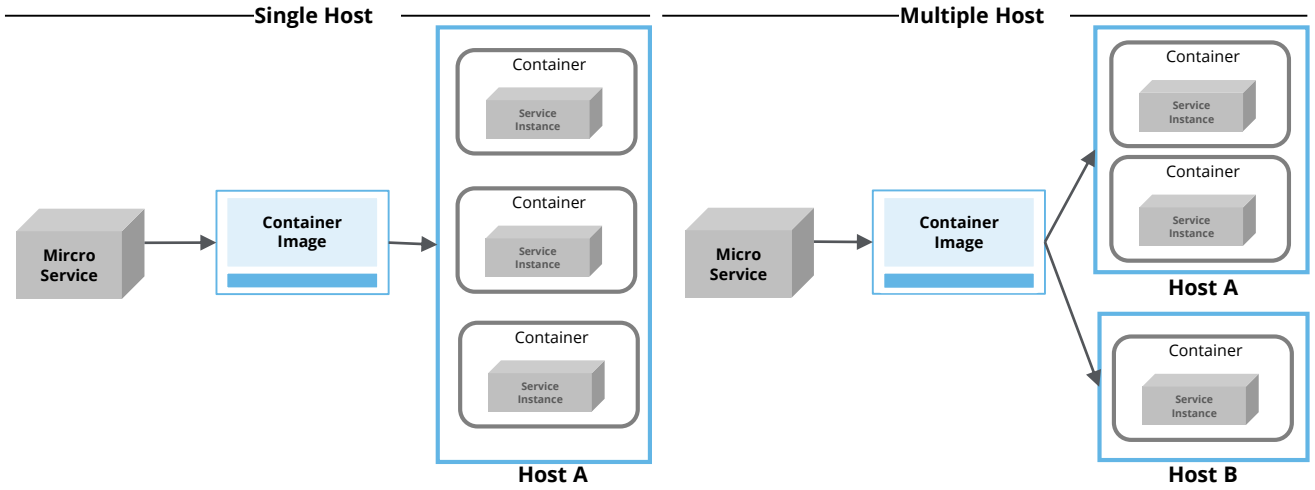
Components of Microservice

- One or more microservices could be added in one container
- Multiple microservices could leverage shared code libraries. However, all the microservcies need to be created and updated as a group.
- Best practice is to deploy only one microservice in one container



Sample Microservice Container Implementation

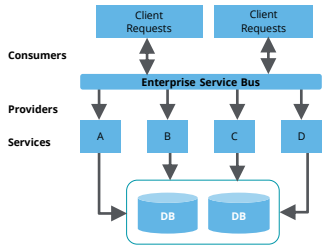
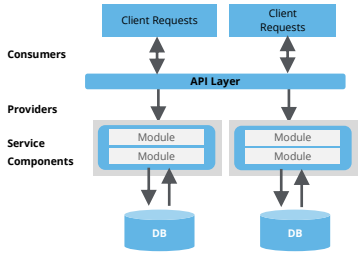
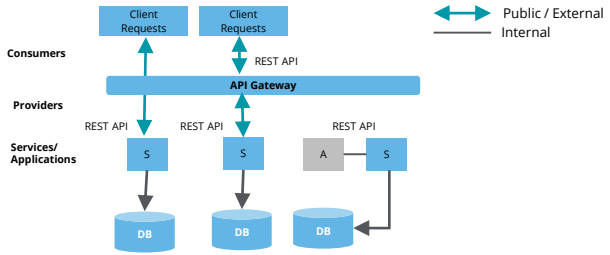
- Each service comprises of multiple service instances and must be isolated from one another
- Package each service as a container image and then deploy each service instance as a container
- Service instances of the micro service could be deployed in one host or multiple hosts
- Each service instance behavior needs to be monitored for availability and performance
- Container orchestrations tools such as Kubernetes, Docker Swarm, Mesos, etc. can be used to effectively manage the containers and microservices at scale



Note: Implementation methods of microservcies are not limited to the one shown above

Demystifying SOA, Microservices and APIs

SOA and Microservices are two different architecture patterns that decomposes an application into services, whereas APIs provide an interface to access these services

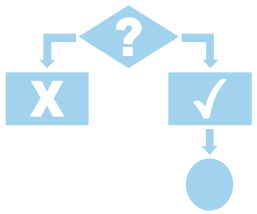
	SOA	Microservices	APIs
Definition	In Service Oriented Architecture, functions of applications are exposed through services to different components via a common interface	In Microservice Architecture, applications are divided into several lightweight services responsible for running a specific business functionality	Application Programming Interfaces is an interface created for services or components of an applications to enable communication
High-Level Architecture			
Communication Scope	Applications can communicate with another by accessing services through common communication bus i.e. Enterprise Service Bus	Each application is independent and scope of communication is only within the application	APIs can be used to expose services internally within the enterprise (Enterprise API) and externally (Public APIs)
Storage	Each services share the data storage	Each services can have an independent data storage	Create APIs to access database servers
Deployment	Services of an application are deployed at once	Each service of an application can be deployed independently	APIs are well documented and open in nature and are often capable of self-provision
Fault Tolerance	Services communicate through Enterprise Service Bus and hence it becomes a single point of failure	Each microservice is independent of the another and does not affect other microservices in case of failures	APIs are dependent on the underlying applications and services
Design Patterns	Service Bus, Event Driven, etc.	Aggregator, Proxy, Chained, Branch, Asynchronous Messaging, etc.	Web Service, Pragmatic REST(URI), Hypermedia and Event Driven
Governance	Centralized, services are governed from design to deployment which also includes policy enforcement. Each service is built on common governance and standards.	Decentralized, no common standard required for designing and developments of services	API management platform may be required to manage API's on a large scale
When to use	Systems where services are shared across the applications and functions	Small systems with minimum services being shared across the applications and functions	Systems where services need to interact and exchange information with each other

Because serverless functions are so new, the **understanding and maturity of their use** and the tools surrounding them are still evolving.

Serverless computing is a model of IT service delivery in which the underlying enabling resources are used as an opaque, virtually unlimited, shared pool that is continuously available without advance provisioning (pre-provisioning) and priced in the units of the consumed IT service

Typical Function as a Service

Event Source

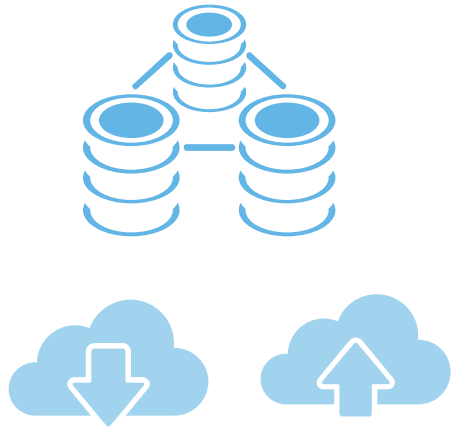


- Change in Data State
- Request to End points
- Change in Resource State

Function

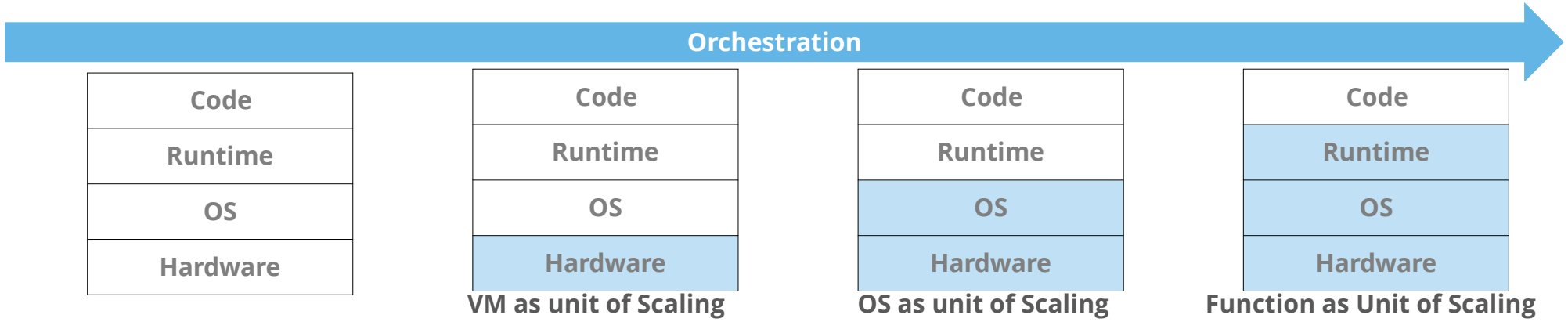
- Node
- Java
- Python
- C#

Services (Anything)



Serverless Architecture (FaaS) – Evolution

Over time, orchestration of Compute services evolved from Virtual Machine as as unit of scaling to Function/Code as a unit of Scale



	Physical Servers	Virtual Machines	Containers	Serverless
Deployment	Slow-iteration and deployment	Faster-iteration and deployment	Fastest-iteration and deployment	Rapid iteration and deployment
Tenancy	Single tenency	Multi-tenency	Super multi-tenancy	Extreme multi-tenancy
Polyglot Friendliness	Unfriendly	Somewhat friendly	Friendly	Very friendly
Provisioning Time	Deploy in weeks	Deploy in minutes	Deploy in seconds	Deploy independently
Tenure	Typically alive for years	Typically alive for weeks	Typically alive for hours	Typically alive for seconds

Serverless Architecture (FaaS) – Use Cases

Functions tend to lend themselves more readily to use cases that have highly variable scaling requirements, integrate or extend other services, and do not have rigorous response-time requirements.

Batch Processing	**Most scheduled processing needs** — such as extraction, transformation and loading (ETL) jobs and MapReduce processing — due to their noninteractive nature and whether they are **short-run.**
Microservices	Microservices rely on isolated application units that leverage "shared nothing" architecture to scale horizontally. The **rapid scalability of serverless technologies** is particularly suited to a microservices architecture.
IT Automation and Integration	Due to its speed, serverless computing can play a significant role in IT automation, **especially for automation tasks that have a particular trigger or event** (for example, tasks such as patching or backup). FaaS is inherently event-driven, it can quickly identify triggers and respond to events
Stream Processing	Serverless computing is an ideal candidate for stream processing is characterized by **high data ingest rates and unpredictable traffic patterns** that require real-time processing. (edge processing, as well as for artificial intelligence (AI)-based learning environments and voice-enabled stateless processing from devices such as Alexa or Google Home)
Noninvasive Extension	As functions can be invoked by events, it is natural to have them execute in response to events that other applications generate, even though those other applications predate or were not designed with the functions in mind, **existing applications — even those for which the source code is not available — can be extended.**
Code-as-Content	Architectures similar to the Amazon Alexa **enable developers to provide additional functionality through publishing their functions to marketplaces and repositories,** effectively treating these extensions as content.

Source: Gartner

Serverless Architecture (FaaS) - Limitations

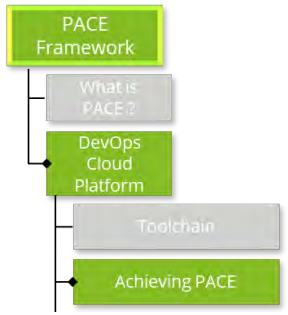
As a nascent ecosystem, few serverless framework products have been in general availability only for an year, most of the vendors offer a proprietary solutions

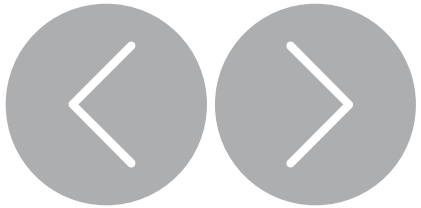
Vendor Lock-in	There are two levels of vendor lock-in that customers should be wary of — ▪ operational tooling and products to execute and manage the code ▪ changes to the code itself due to varied programing frameworks that different providers support
Lack of Choice in Operating Tooling	▪ Plenty of operational know-how and tooling is required in the areas of security, monitoring and debugging. ▪ In most cases, customers will be forced to use the tools provided by the platform provider, which may not be a best-of-breed tool with in-depth capabilities.
Narrow Use Case Fit	▪ Only for driven computing use cases, where the runtime of the code is minimal, FaaS works well. ▪ Unlike VMs and containers that can hold resources for hours or weeks, functions are built to execute code for seconds or milliseconds.
Loss of Control Over Servers	▪ Serverless frameworks do not provide a mechanism for customizing your computing environment to suit the needs of your workload. ▪ Functions mostly operate in a stateless manner on the host. Where state is maintained, it is typically in an external database and storage system, which induces latency issues for stateful applications.

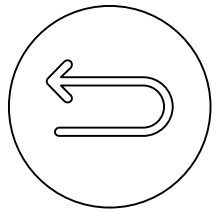
Source: Gartner

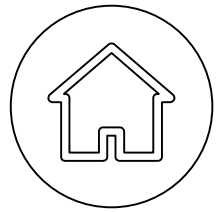
How to use the playbook

Access the information you need through the left navigation bar, TABs and buttons at the top of each page and hyperlinks from within slide content.









The clickable navigation bar on the left-hand side indicates the section or page you are in

The scroll buttons help:

- move back/ up a level and,
- move forward for continuing topical content

When viewing this presentation mode in Microsoft PowerPoint, the return button takes you back to the last slide/ topic you were on

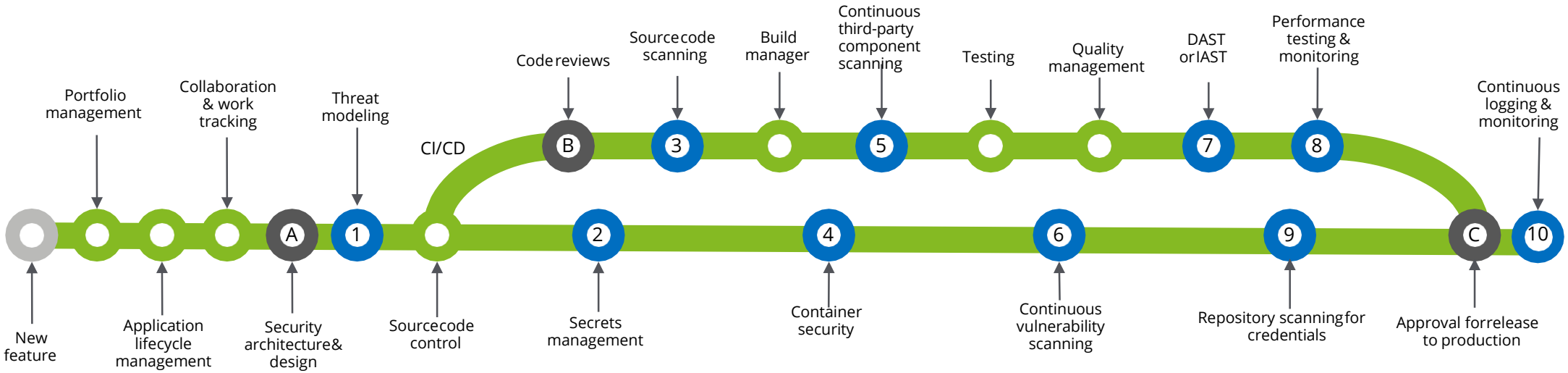
The home button takes you back to the homepage

Deloitte.

#	Term	Definition
1	Driver and Navigator in Pair Programming	The programmer at the keyboard is usually called the driver. The other person, the navigator, is also involved in the programming task, focusing on the overall direction. The two programmers regularly swap roles.
2	Test-Driven Development	Test-driven development involves 1) Writing a failing test; 2) Writing the simplest of code to pass the test; and 3) Refactoring the code to remove duplication.
3	Behavior-Driven Development	Behavior-driven development makes use of a simple, domain-specific scripting language that converts structured natural language statements into executable tests.
4	Microservices Architecture	This is a method of developing software applications as a suite of independently deployable, small, modular services in which each service runs a unique process and communicates through a well-defined, lightweight mechanism to serve a business goal.
5	Requirements Traceability	Requirements traceability refers to the ability to link product requirements from the stakeholders' rationales and to the corresponding design artifacts, code, and test cases.
6	Automated Environment Configuration	Server setup steps can be translated into a set of provisioning scripts, which can then be checked-in to version control (similar to software source code) to track changes.
7	System Architecture	System architecture patterns like microservices or service-oriented architecture enable teams to be independently productive and sufficiently decoupled from each other.
8	Static Code Analysis and Code Coverage	Static code analysis is a method of computer program debugging that involves examining the code without executing the program. Code coverage is a measure that should be used to describe the degree to which the source code of a program is executed when a particular test suite runs as part of continuous integration builds.
9	Performance Tests and Other Non-Functional Tests	Performance Tests: Tests that validate performance across the entire application stack (code, database, storage, network) and are part of the deployment pipeline so the problems are detected early and fixes are cheaper and faster. Other Non-Functional Tests: Tests that validate non-functional requirements (e.g., availability, scalability, capacity, security), which are typically fulfilled by the correct environment configurations, supporting databases, libraries, and other dependencies. These are also a part of the deployment pipeline.
10	Cluster Immune	The cluster immune pattern builds on the canary strategy to link production monitoring system with the release process by automating a code rollback when user-facing performance of the production system breaches the pre-defined acceptable range (e.g., conversion rate for new users drop below historical norm of 15-20%).
11	Artifact Repository	An artifact repository is used to store “release” artifacts such as War or Ear files, which are binary files. as opposed to source code, which are ASCII files.
12	Refactoring	Refactoring refers to the iterative (through a sequence of small changes) restructuring of an existing body of code, altering its internal structure without changing its external behavior.
13	Pair Programming	Pair programming involves two programmers (driver and navigator) sharing a single workstation.

Illustrative DevSecOps process flow

DevOps Processes and capabilities across the value stream



Color Legend

Green	DevOps Processes
Blue	Security Automation
Gray	Security Processes

Activity No.	DevSecOps Activity	Priority (cyber perspective)	Process
A	Security architecture and design	Required	Manual
B	Code reviews	Required	Manual
C	Approval for release to production	Required	Manual via automated workflow
1	Threat modeling	Recommended	Manual or tools based
2	Secrets management	Required	Automated
3	Source code scanning	Required	Automated
4	Container security	Required (if used)	Automated
5	Continuous third-party component scanning	Required (if used)	Automated
6	Continuous vulnerability scanning	Required	Automated
7	Dynamic Application Security Testing	Required	Automated
8	Performance testing & performance monitoring	Recommended	Automated
9	Repository scanning for credentials	Required	Automated
10	Continuous monitoring & logging	Recommended	Automated