DevOps Automation

Service Catalogue

CONTENTS

AEM Background

AEM Corporation is a diversified services company that primarily supports federal agencies and Fortune 1000 clients. We employ leading experts in information technology; cybersecurity; data management and analysis; research, development, and evaluation; engineering; technical assistance; and operations management. Founded in 1986, we have leveraged these strengths to become one of America's fastest-growing companies. Learn more at aemcorp.com.

DevOps Expertise

AEM works with complex applications on behalf of private-and public-sector clients, tailoring support to their needs. We ensure measurable outcomes by offering substantial experience in software system development lifecycles, backed by leading qualifications in cloud, container, and CI/CD technologies.

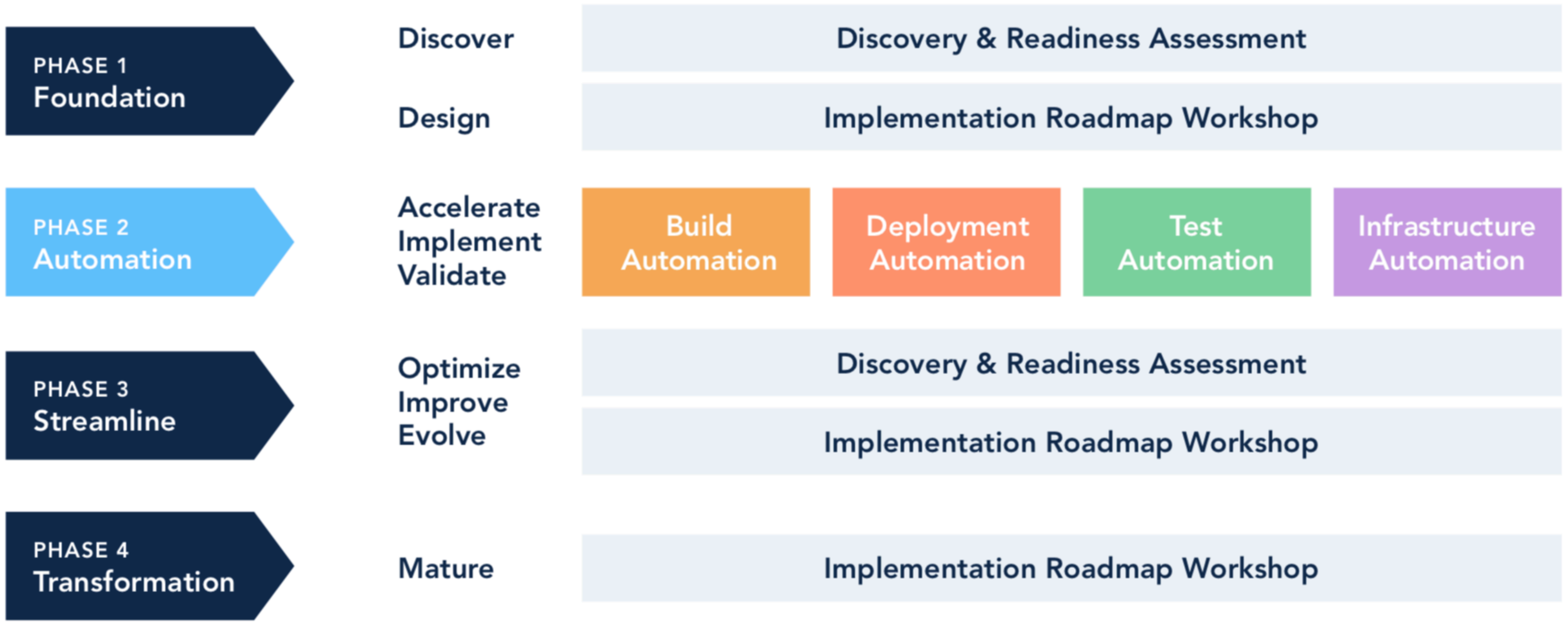
AEM experts have deep expertise with the tools across the DevOps ecosystem that are essential to promoting a collaborative project environment. As such, we accelerate your initiatives by integrating DevOps tools and processes from the start, and we then sustain them by nurturing a continuous learning environment for your community of DevOps practitioners.

DevOps Automation

Just as the Industrial Revolution introduced the manufacturing industry to rolling assembly lines and automated processes, DevOps processes are ingrained with automation techniques that provide rapid feedback, repeatable processes, and consistent creation of business application systems. With the convergence of multiple systems and IT professionals all focusing on the delivery of these applications, these processes quickly become complex and can seem daunting to implement for the first time.

Our experts have grouped these processes into four areas of focus for implementing automation that can each help your organization gain efficiencies and improve your overall IT processes: Build, Deployment, Test, and Infrastructure. With each of these areas, AEM's experts will guide your organization in the development of optimized IT processes that adopt core enabling disciplines, monitor key performance indicators, and avoid potential red flag areas.

Our Pragmatic DevOps Services Framework



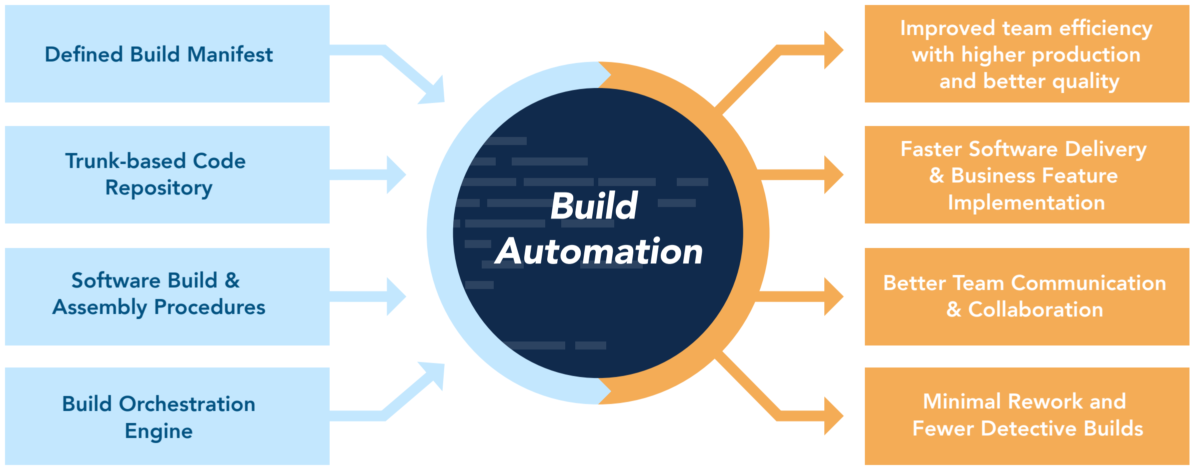
BUILD AUTOMATION

Build Automation is the process of scripting and automating the retrieval of software code from a repository, compiling it into a binary artifact, executing automated functional tests, and publishing it into a shared and centralized repository.

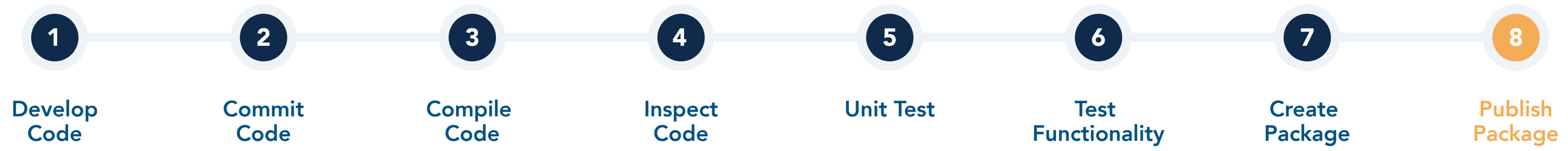
- Define and execute a consistent and repeatable process.
- Amplify feedback and improve team communication.
- Improve overall release deployment quality.
- Accelerate the implementation of desired business features/functionalities.

Models

In this diagram, given the items on the left, implementing Build Automation will result in the items on the right.


Defined Build Manifest
Trunk-based Code Repository
Software Build & Assembly Procedures
Build Orchestration Engine
Build Automation
Improved team efficiency with higher production and better quality
Faster Software Delivery & Business Feature Implementation
Better Team Communication & Collaboration
Minimal Rework and Fewer Detective Builds

A typical automated build process is shown in the diagram below. This will vary slightly based upon team structure and organization focus.



Red Flag Areas

Lack of Source Code Tool Expertise is a barrier to frequent code commits. This often leads to complicated integration issues and inhibits overall team visibility of the code base.

Complicated Source Code Branching leads to duplicate changes and complex merges.

Infrequent Check-ins of Code leads to complex change merges and delayed integrations.

Broken Builds diverts focus of development teams away from new feature creation.

Minimal Feedback negatively affects overall quality due to "lack of eyeballs."

Excessive Notifications result in team members ignoring important alerts.

Bloated Builds prevent rapid feedback and creates potential bottlenecks.

Infrequent Builds delay feedback and identification of potential integration issues.

Key Metrics

Number of Features / User Stories per Build

indicates the number of changes being implemented and maps to business value being created.

Change Implementation Lead Time

affects the number of releases per a given period and overall product roadmap planning.

Average Build Time

indicates the number of changes being implemented and maps to business value being created.

Frequency of Builds

indicates the overall output and activity of the project.

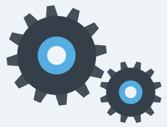
Percentage of Failed Builds

impacts the overall team output due to rework.

Key Roles

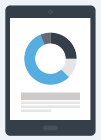
Developer

Responsible for taking business requirement and implementing in software code



Build Engineer

Responsible for defining the release pipelines and maintaining the build infrastructure



Requirements Analyst

Responsible for defining the user story to include the definition of acceptance criteria

Key Steps

1 Assess & Inventory

Survey the current software build process and tools and identify key goals to accomplish. Focus on source code repositories, build manifests, assembly processes, and artifact repository.

2 Design & Model

Identify a build and assembly process and toolchain that will support the creation of the software artifacts.

3 Configure & Build

Set up the Build Automation toolchain to include the implementation of triggers upon source code commit and publishing of software artifacts.

4 Deploy & Validate

Integrate the Build Automation toolchain and orchestration processes with an existing project. Validate that the build is complete with clearly defined stages that can be measured.

5 Measure & Manage

Capture the metrics related to the build process to include build times, successful vs. failed builds, code quality, and number of published versioned artifacts.

Related Process Focus Areas

Test Automation

Improving the quality and completeness of test cases and easing the execution and result summary through automation

Infrastructure Automation

Enabling the creation and destruction of server and application infrastructure to support the development and testing processes

Release Automation

Provisioning the application artifacts and configurations to the operating environments across the system development lifecycle

Agile Release Planning

Capturing and refining the requirements to create small units of work that can be implemented into software code

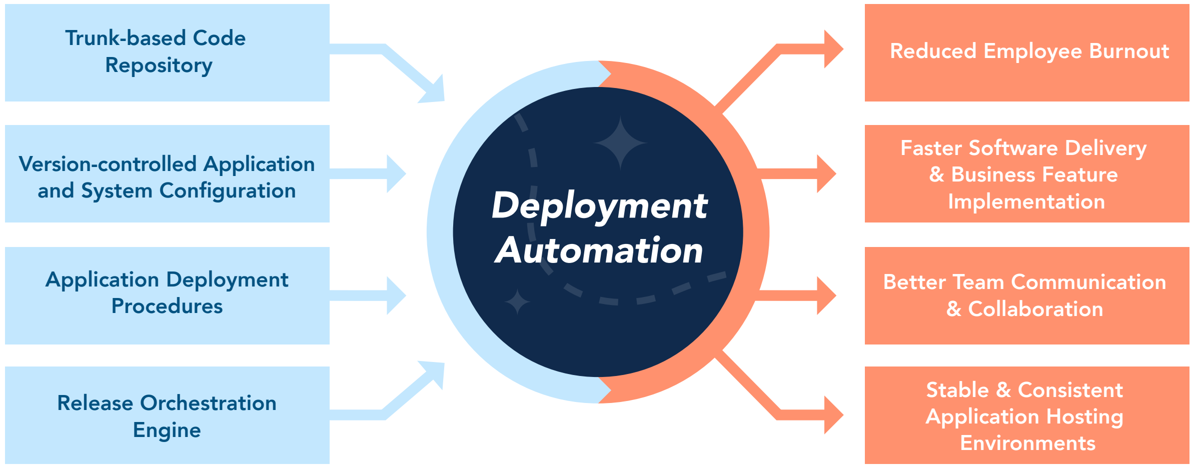
DEPLOYMENT AUTOMATION

Deployment Automation is the process of provisioning the application artifacts and configurations to the operating environments across the system development lifecycle. It entails a combination of application deployment automation, environment modeling, and workflow orchestration to achieve rapid delivery of application features in a reliable and orderly manner.

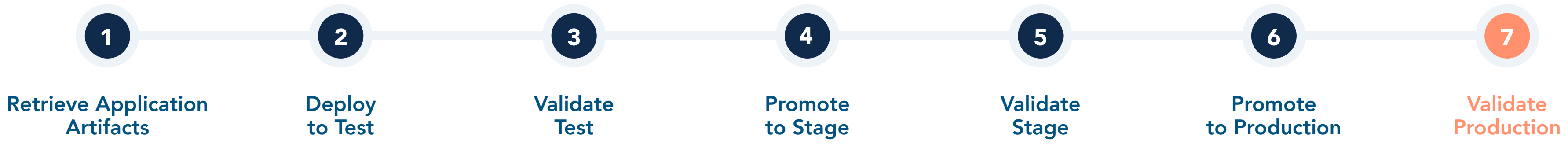
- Define and execute a consistent and repeatable process.
- Improve overall release deployment quality.
- Amplify feedback and improve team communication.
- Accelerate the delivery of application features to production.

Models

In this diagram, given the items on the left, implementing Deployment Automation will result in the items on the right.


Trunk-based Code Repository
Version-controlled Application and System Configuration
Application Deployment Procedures
Release Orchestration Engine
Deployment Automation
Reduced Employee Burnout
Faster Software Delivery & Business Feature Implementation
Better Team Communication & Collaboration
Stable & Consistent Application Hosting Environments

A typical release process is shown in the diagram below. This will vary slightly based upon team structure and organization focus.



Red Flag Areas

Lack of Source Code Tool Expertise is a barrier to frequent code commits. This often leads to complicated integration issues and inhibits overall team visibility of the code base.

Embedded System and/or Application Configurations often require complicated code changes to account for specific environment configurations and may introduce complex code merges for different target environments.

Infrequent Check-ins of Configurations lead to misconfigured environments and tribal knowledge of environment requirements.

Inconsistent Environments increase deployment tasks and introduce instability into environments due to variations in topology and configurations.

Excessive Notifications result in team members ignoring important alerts.

Long-running Deployments prevent rapid feedback and create potential bottlenecks.

Infrequent Deployments lead to the outdated system provisioning procedures and environment definitions, which ultimately impact the overall system stability.

Key Metrics

Time to Fulfill Environment Provisioning Request

redirects resources from application and business feature development and impacts time required for each release.

Average Deployment Time

impacts the available time for deployments.

Number of Features/ User Stories per Build

indicates the number of changes being implemented and maps to business value being created.

Percentage of Failed Deployments

impacts the overall team output due to rework.

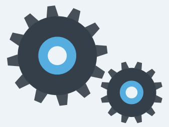
Frequency of Deployments

indicates the overall output and activity of the project.

Key Roles

System Administrator

Responsible for preparing and modeling the target environments and deploying the application into each of them



Build Engineer

Responsible for defining the release requirements and supporting the application deployment

Release Manager

Responsible for defining and communicating the application release plans and reporting on the progress of the deployments

Key Steps

1 **Assess & Inventory**

Survey the current environment provisioning process and tools and identify key goals to accomplish. Focus on external system and application configurations, application deployment procedures, and release deployment orchestration.

2 **Design & Model**

Define the target application environments to include the environment-specific system and application configurations.

3 **Configure & Orchestrate**

Set up the Build Automation toolchain to include the integration with a central application artifact repository and target environments.

4 **Deploy & Validate**

Integrate the release automation toolchain and orchestration processes with an existing project. Validate that the deployment is complete with clearly defined stages that can be independently managed.

5 **Measure & Manage**

Capture the metrics related to the deployment process to include deployment times, successful vs. failed deployments, environment consistency, and number of managed released artifacts.

Related Process Focus Areas

Test Automation

Improving the quality and completeness of test cases and easing the execution and result summary through automation

Infrastructure Automation

Enabling the creation and destruction of server and application infrastructure to support the development and testing processes

Build Automation

Provisioning the application artifacts and configurations to the operating environments across the system development lifecycle

Agile Release Planning

Capturing and refining the requirements to create small units of work that can be implemented into software code

TEST AUTOMATION

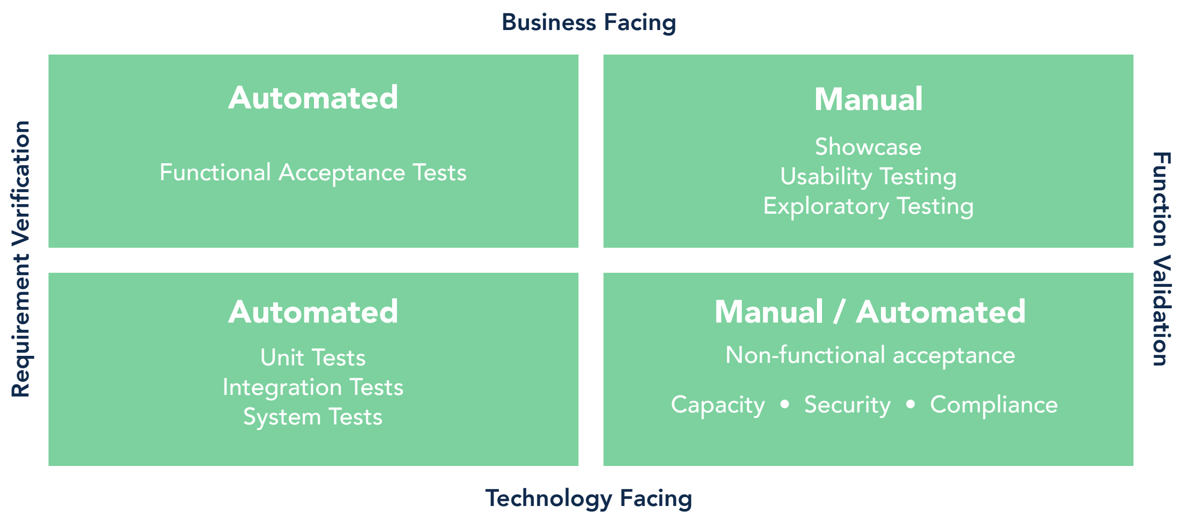
Test Automation is a practice where application tests are run automatically and continuously throughout the development process. Test-driven development and the use of unit tests are used to create and maintain acceptance tests that are reproducible and executed with each build.

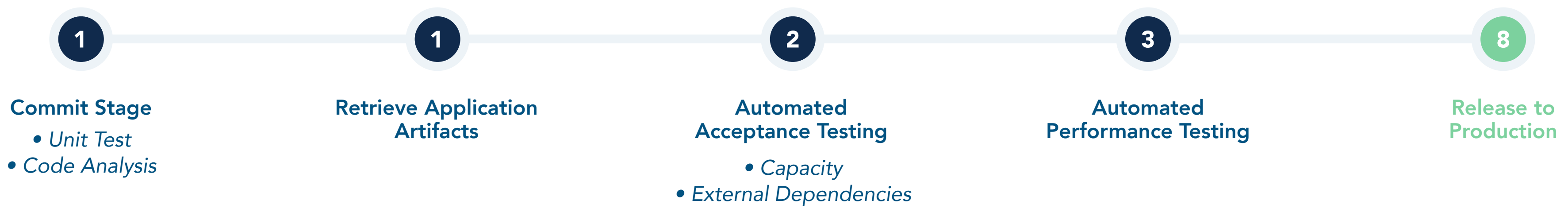
- **Define and execute a consistent and repeatable process.**
- **Improve overall release deployment quality.**
- **Amplify feedback and improve team communication.**
- **Accelerate the delivery of application features to production.**

Models

Applications require several techniques and levels of testing to meet quality and user expectations. A typical project will implement a strategy similar to that of the Agile Testing Quadrants model shown here.

The stages of Test Automation are shown in the diagram below. These will vary slightly based upon team structure and organization focus.





Red Flag Areas

Lack of Code Coverage Statistics
indicates limited testing being done during development.

Lack of Quality Test Data
indicates poor test data management, which will produce unexpected application results in production.

Long-running Test Suite
conflicts with the need for fast builds and often results in tests being skipped during builds.

Excessive Notifications
result in team members ignoring important alerts, particularly for test results that produce false positives for defects.

Key Metrics

Test Code Coverage

identifies the percentage of code in which functionality has been verified.

Defects Reported Post Release

indicates that requirements may not have a common understanding and/or automated testing is incomplete.

Average Test Execution Suite Time

impacts the available time for builds.

Key Roles

System Administrator
Responsible for preparing and modeling the test environments and deploying the application into each of them

Test Engineer
Responsible for defining the release pipelines and maintaining the build infrastructure

Developer
Responsible for the application development and defect resolution

Release Manager
Responsible for defining and communicating the application release plans and reporting on the progress of the deployments

Key Steps

1

Assess & Inventory

Survey the current suite of tests to include code coverage analysis. Evaluate code complexity and identify potential refactorings to improve code testability.

2

Design & Model

Define and identify the test cases and overall suite for functional and user acceptance testing. For each user story, ensure that acceptance criteria are defined and documented.

3

Configure & Orchestrate

Set up the Test Automation tool suite to include the integration with a release pipeline, particularly stages that should be automated.

4

Measure & Manage

Capture the metrics related to the deployment process to include deployment times, successful vs. failed deployments, environment consistency, and number of managed released artifacts.

Related Process Focus Areas

Infrastructure Automation

Enabling the creation and destruction of server and application infrastructure to support the development and testing processes

Deployment Automation

Provisioning the application artifacts and configurations to the operating environments across the system development lifecycle

Build Automation

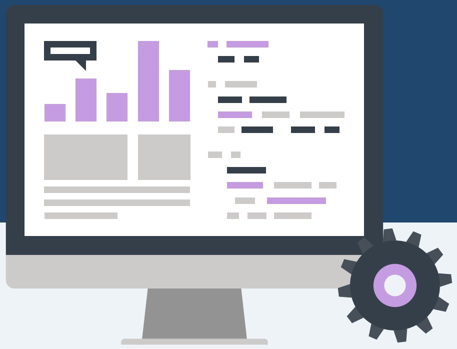
Provisioning the application artifacts and configurations to the operating environments across the system development lifecycle

Agile Release Planning

Capturing and refining the requirements to create small units of work that can be implemented into software code

INFRASTRUCTURE AUTOMATION

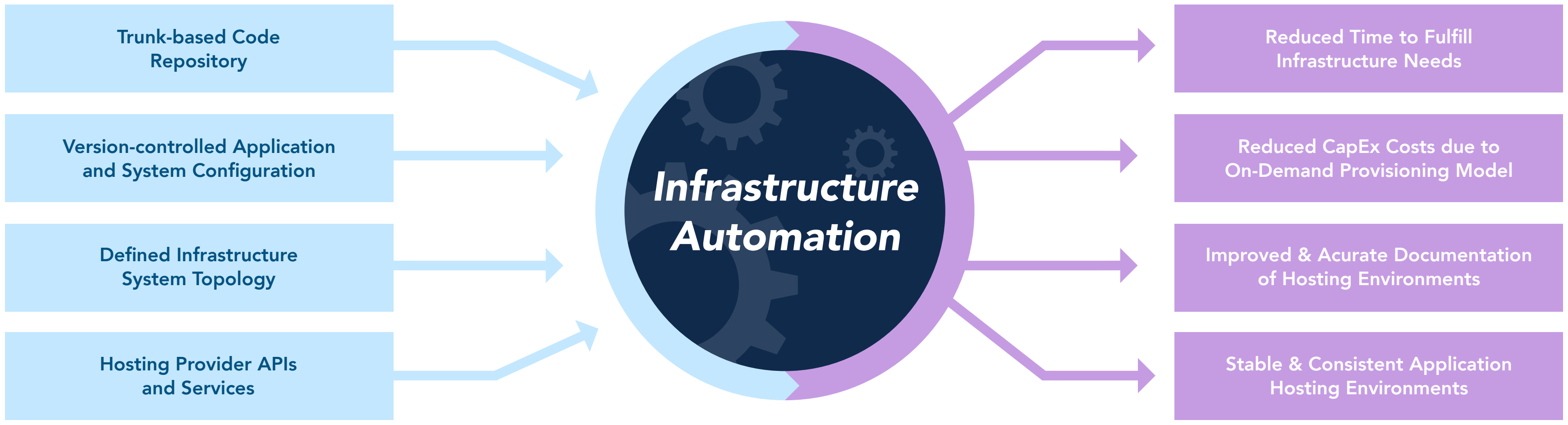
Infrastructure Automation is the process of creating and tearing down server and application infrastructure to support the development, testing, and production environments.



- Define and execute a consistent and repeatable process.
- Amplify feedback and improve team communication.
- Improve overall release deployment quality.
- Accelerate the delivery of application features to production.

Models

In the diagram below, given the items on the left, implementing Infrastructure Automation will result in the items on the right.


Trunk-based Code Repository
Version-controlled Application and System Configuration
Defined Infrastructure System Topology
Hosting Provider APIs and Services
Infrastructure Automation
Reduced Time to Fulfill Infrastructure Needs
Reduced CapEx Costs due to On-Demand Provisioning Model
Improved & Acurate Documentation of Hosting Environments
Stable & Consistent Application Hosting Environments

Red Flag Areas

Embedded System and/or Application Configurations
lead to complicated code changes to handle environment changes and target releases.

Infrequent Check-ins of Configurations
lead to misconfigured environments and tribal knowledge of environment requirements.

Inconsistent Environments
increase deployment tasks and introduce instability into environments due to variations in topology and configurations.

Excessive Notifications
result in team members ignoring important alerts.

Long Provisioning Request Fulfillments
prevent rapid changeover and on-demand creation and scaling of supporting infrastructure elements.

Infrequent Deployments
system provisioning runbooks could atrophy over time and without regular exercising of procedures, environment definitions, which could become outdated.

Key Metrics

Time to Fulfill Environment Provisioning Request

redirects resources from application and business feature development and impacts time required for each release.

Frequency of Deployments

indicates the overall output and activity of the project.

Average Deployment Time

impacts the available time for deployments.

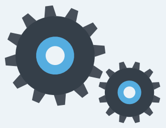
Percentage of Available Capacity

indicates whether an environment is sized correctly and identifies potential stability issues that will occur when available resources are exhausted.

Key Roles

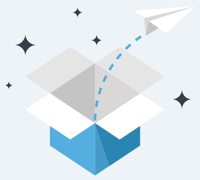
System Administrator

Responsible for preparing and modeling the target environments and deploying the application into each of them



Build Engineer

Responsible for defining the release requirements and supporting the application deployment



Release Manager

Responsible for defining and communicating the application release plans and reporting on the progress of the deployments

Key Steps

1

Assess & Inventory

Survey the current environment provisioning process and tools and identify key goals to accomplish with a focus on external system and application configurations, application deployment procedures, and release deployment orchestration.

2

Design & Model

Define the target application environments to include the environment-specific system and application configurations.

3

Configure & Orchestrate

Set up the Build Automation toolchain to include the integration with a central application artifact repository and target environments.

4

Deploy & Validate

Integrate the release automation toolchain and orchestration processes with an existing project. Validate that the deployment is complete with clearly defined stages that can be independently managed.

5

Measure & Manage

Capture the metrics related to the deployment process to include deployment times, successful vs. failed deployments, environment consistency, and number of managed released artifacts.

Related Process Focus Areas

Infrastructure Automation

Enabling the creation and destruction of server and application infrastructure to support the development and testing processes

Deployment Automation

Provisioning the application artifacts and configurations to the operating environments across the system development lifecycle

Build Automation

Provisioning the application artifacts and configurations to the operating environments across the system development lifecycle

Agile Release Planning

Capturing and refining the requirements to create small units of work that can be implemented into software code