Course Introduction

In this course, you will learn the key principles of DevOps and the value that it can bring to teams that adopt this culture. Key topics that we will discuss in this course:

* What is DevOps
* DevOps Principles
* Various Stages in DevOps Lifecycle
* Metrics to measure the effectiveness of DevOps

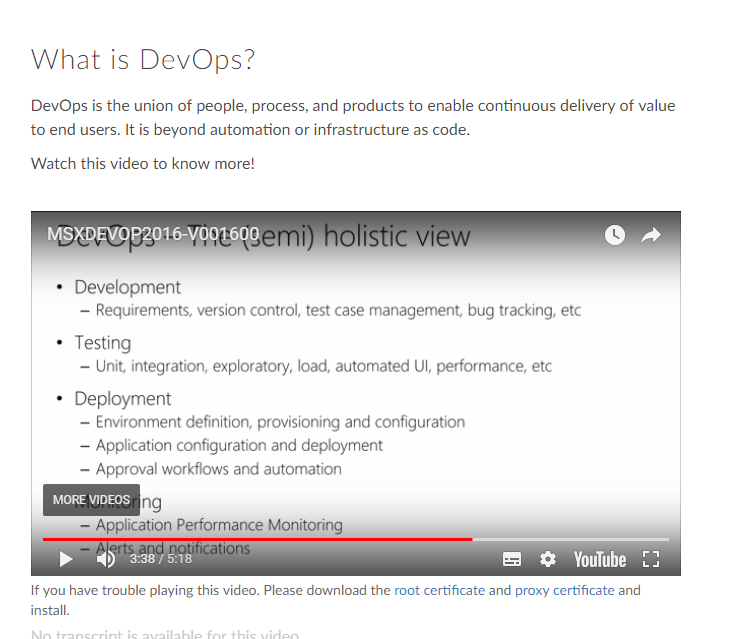
##### What is DevOps?

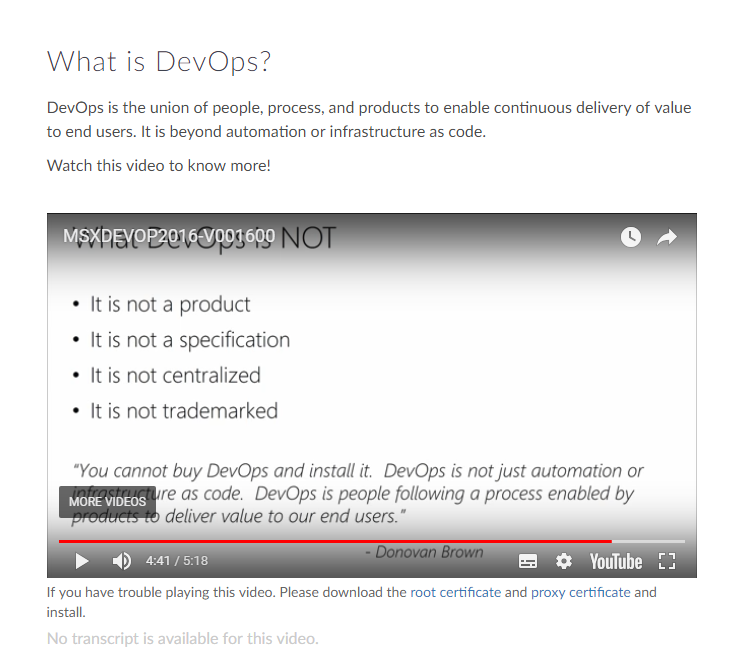
DevOps is the union of people, process, and products to enable continuous delivery of value to end users. It is beyond automation or infrastructure as code.

Watch this video to know more!









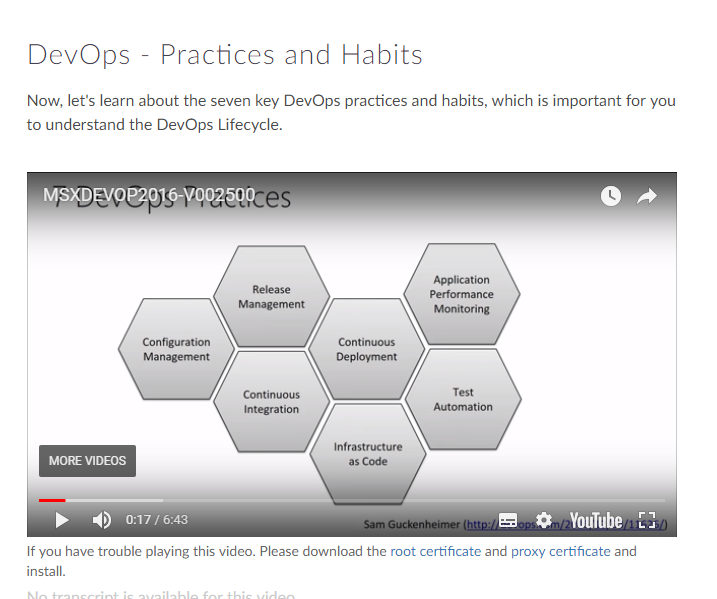
Value of DevOps

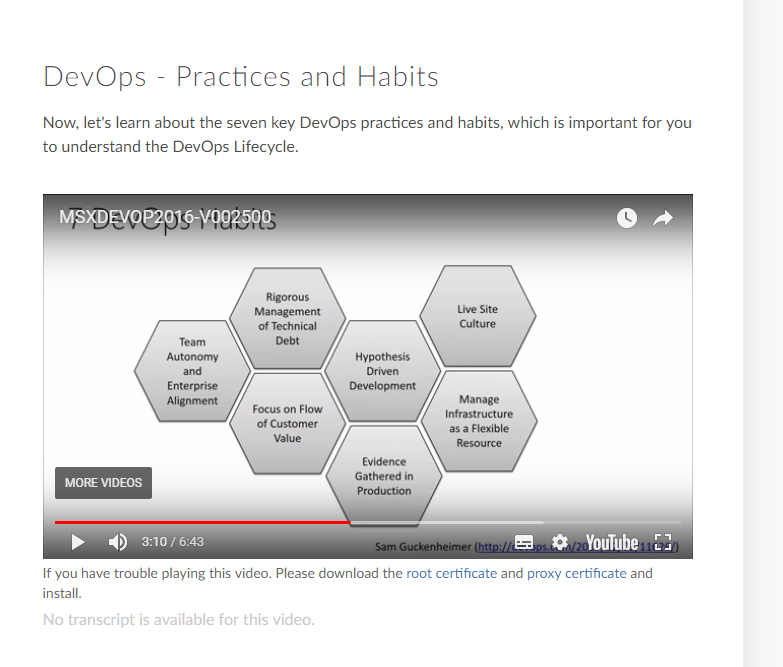
DevOps culture nurtures better communication and collaboration between teams, thus breaking the silos formed among teams.

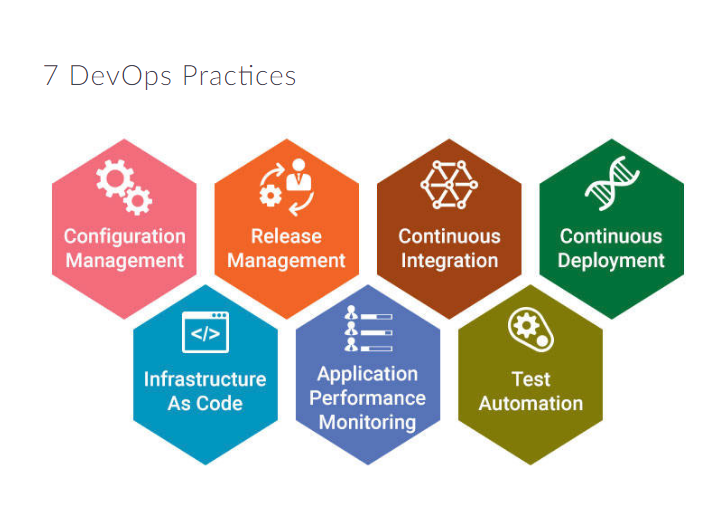
DevOps aims at:

* Developing high quality software
* Deploying in frequent cycles
* Reducing time to move from idea to implementation



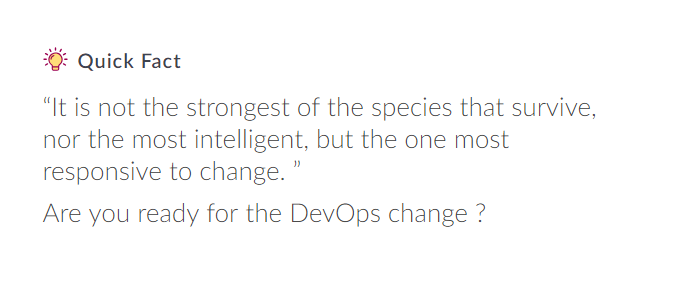












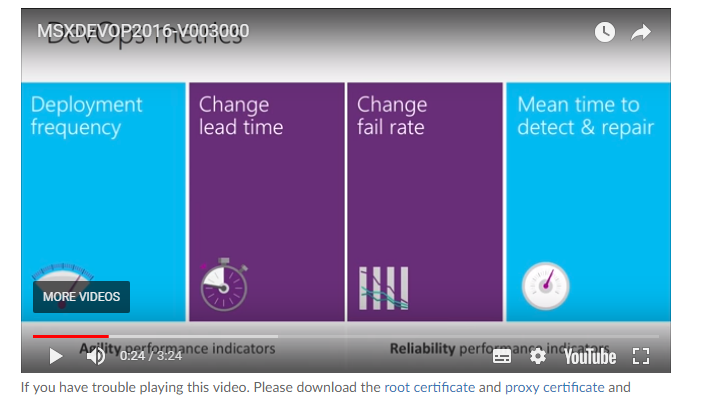
##### Module Outline

In this module, you will understand, KPIs and metrics that are typically analyzed in DevOps culture.

Also, you will learn about the agility performance and reliability performance indicators.

##### DevOps - Metrics

This video will introduce you to the DevOps Metrics Agility performance indicators and Reliability performance indicators that are typically collected in DevOps.



Metric - Deployment Frequency



***Deployment Frequency*** is a direct or indirect measure of

* Response Time
* Team Cohesiveness
* Developer Capabilities
* Effectiveness of Development tool
* Efficiency of the team

Ideally Deployment Frequency should show an upward trend or remain stable week on week.

##### Metric - Change Lead Time



**Change Lead Time** measures complexity of the code and the efficiency of the development systems.If the change lead time is too long, it may be an indication that the development/deployment process is inefficient in certain stages, or that it includes performance bottlenecks.

##### Quick Fact

“In Ops you can never exceed expectations, because the expectation is 100% up-time”

##### Metric - Reliability Performance



**Change Failure Rate** is a good measure of the value of deployments in DevOps.

A low failure rate is a good indication of a healthy DevOps team

**Mean Time To Recover (MTTR)** is a measure of failure rate. This can be seriously affected by code complexity, number of additional features being added and environmental changes.

##### Source Control in DevOps

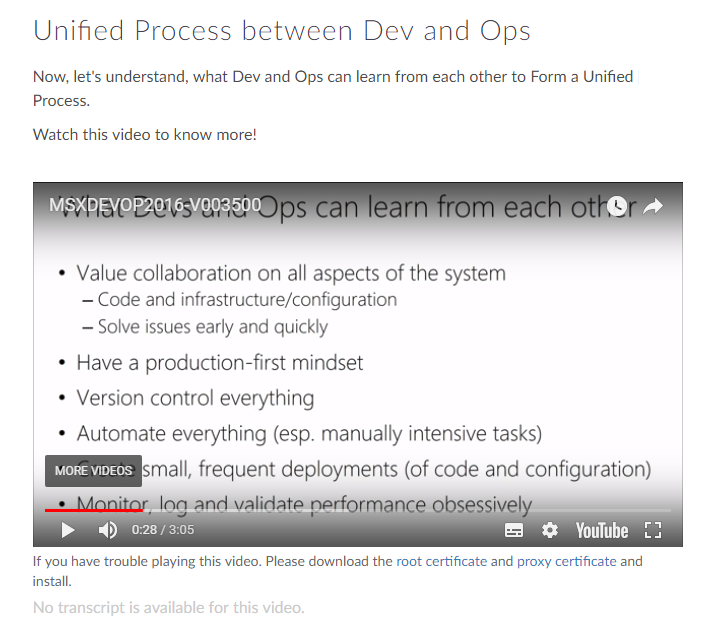
In DevOps, we treat infrastructure as code. Hence version or source control is an integral part of the culture.

**The ultimate goal is to treat the infrastructure code in the same lines on the application code**.

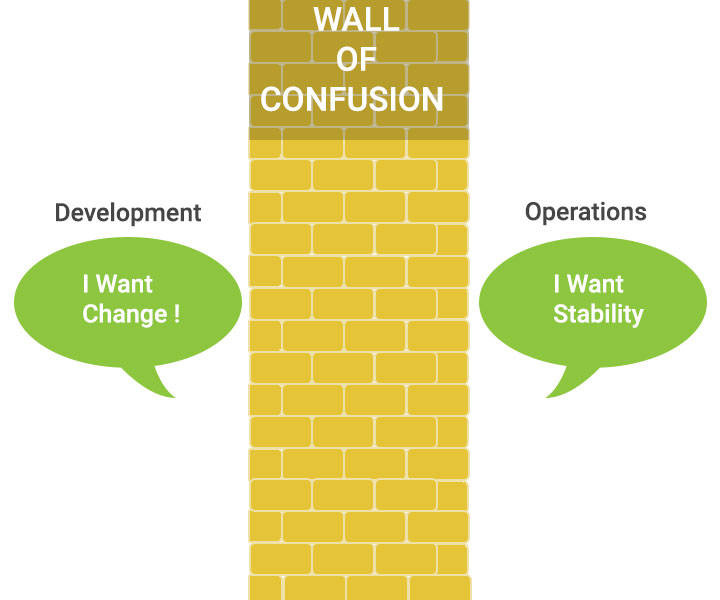
Watch this video to learn more about importance of version control in DevOps.

##### Unify Dev and Ops

In this module, you will learn how to deliver incremental value and the importance of communication and collaboration in Unifying Dev and Ops.



Collaboration and Communication



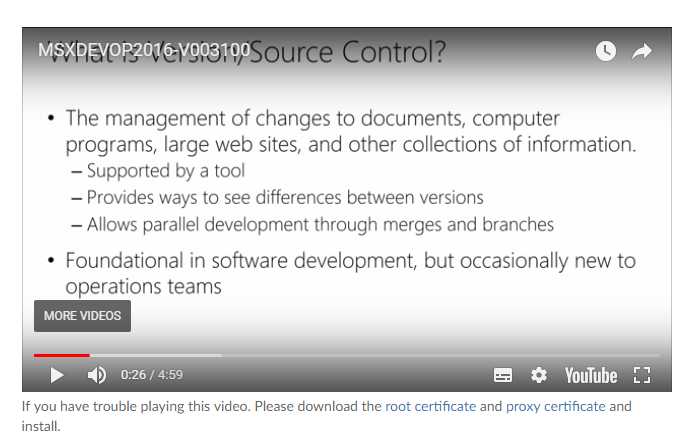
Collaboration and Communication are essential for the success of DevOps team. Fostering a culture of Collaboration and Communication helps:

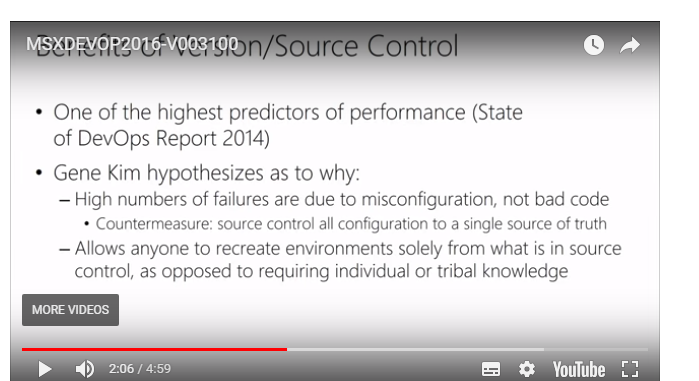
* Break barriers among team members
* Create visibility

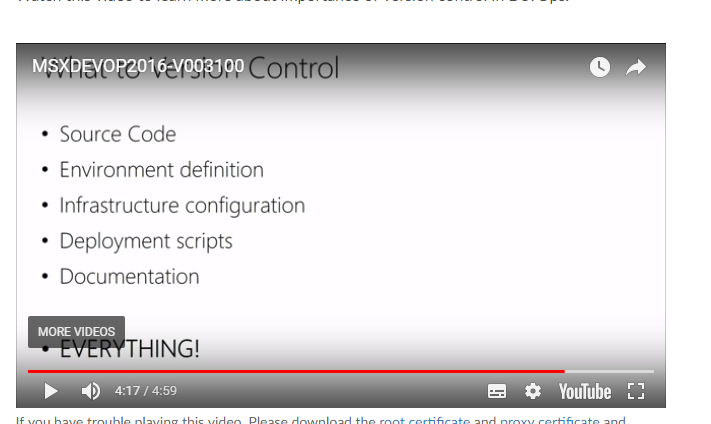
##### Quick Fact

“Devs are from Venus, Ops are from Mars”

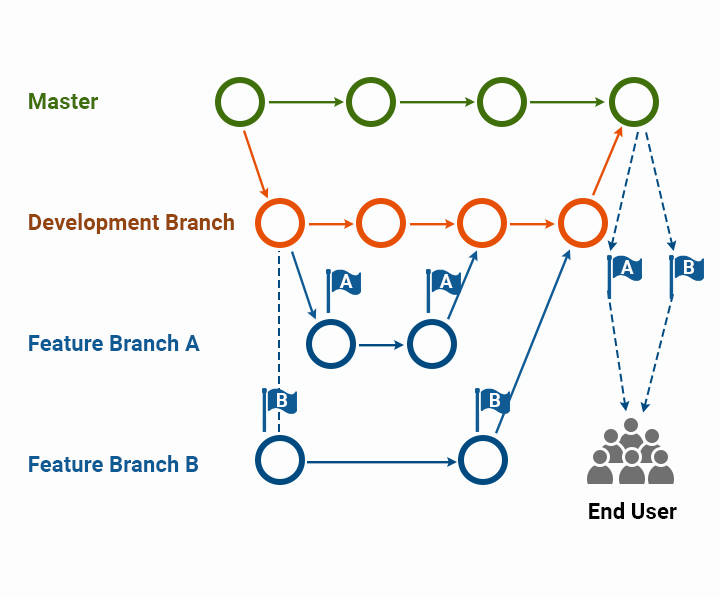
Time to break the Wall of Confusion!







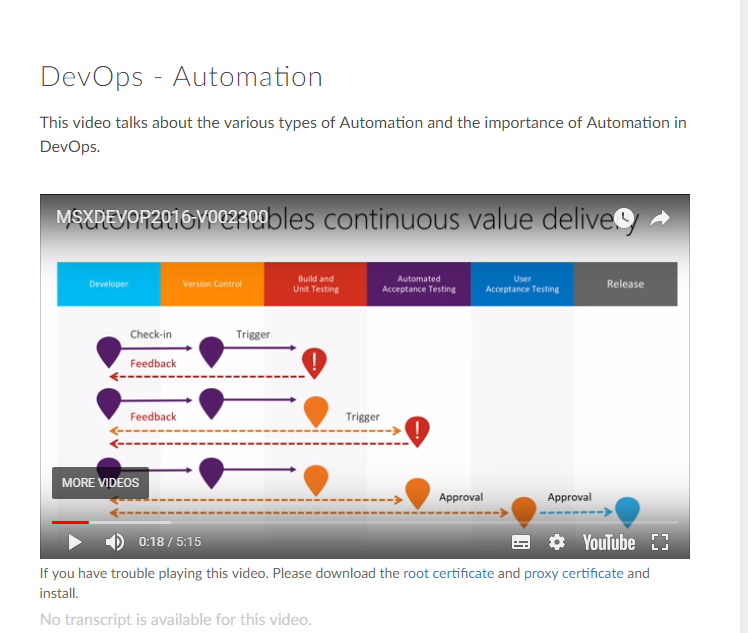
##### Branching Illustrated

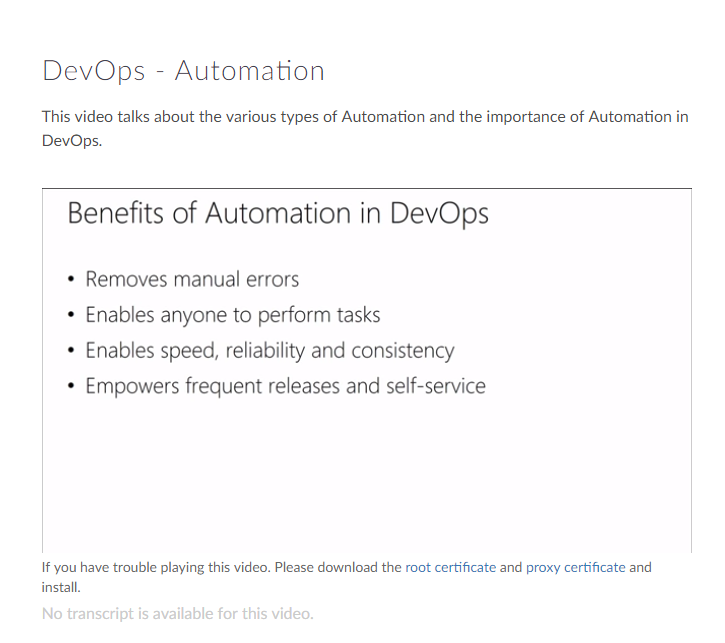


Branching Strategy

Ideally your branching strategy should accomplish the following three goals:

* Minimize conflicts when merging code
* Track code changes in the development pipeline
* Add a degree of separation between code at different stages of development







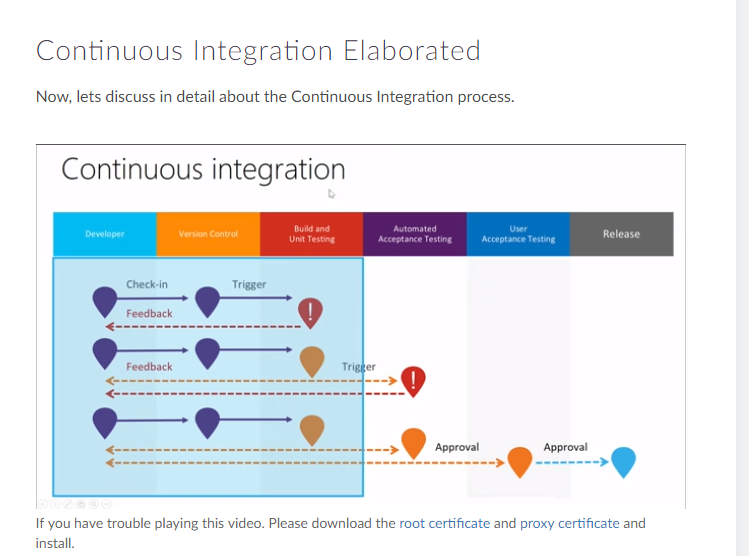
Automation Recommendation

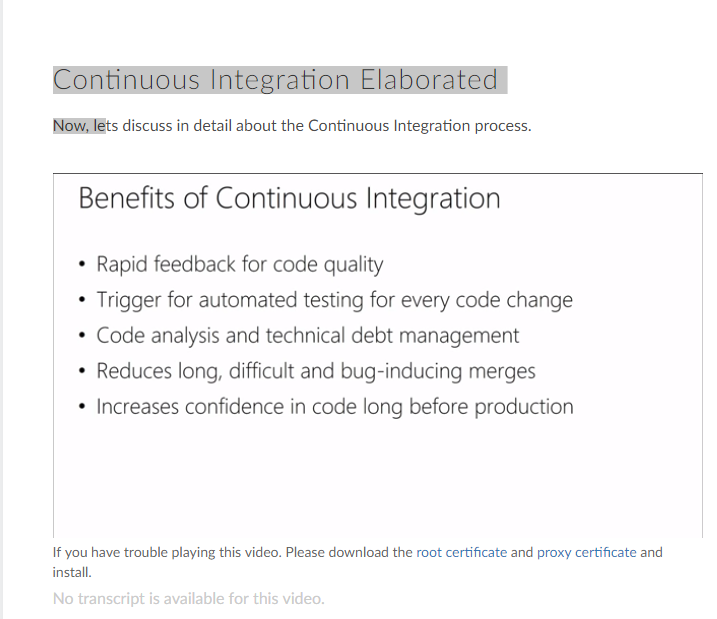
* It is recommended not to implement full automation. i.e. automate just what is necessary.
* Automation is not an independent discipline in itself.
* Automation is done to obtain fast feedback.

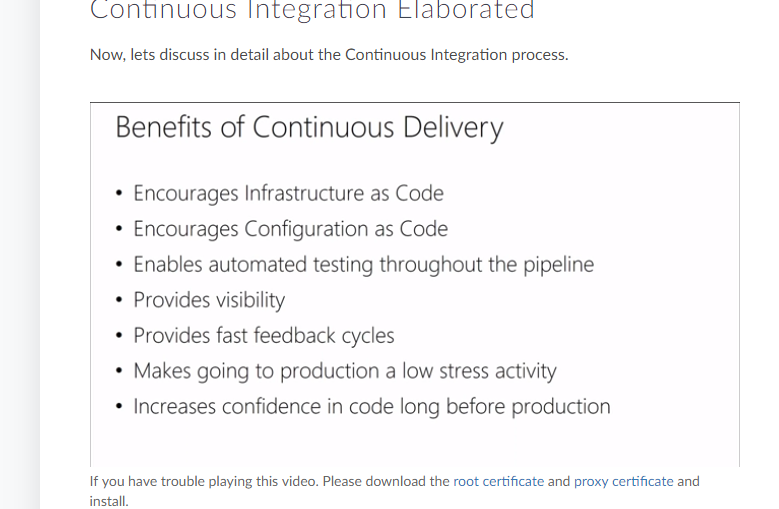
##### Continuous Integration

**Continuous Integration** forces the developers to integrate their code at an early stage which prevents integration issues from happening towards the end of the lifecycle.









Benefits of Continuous Integration

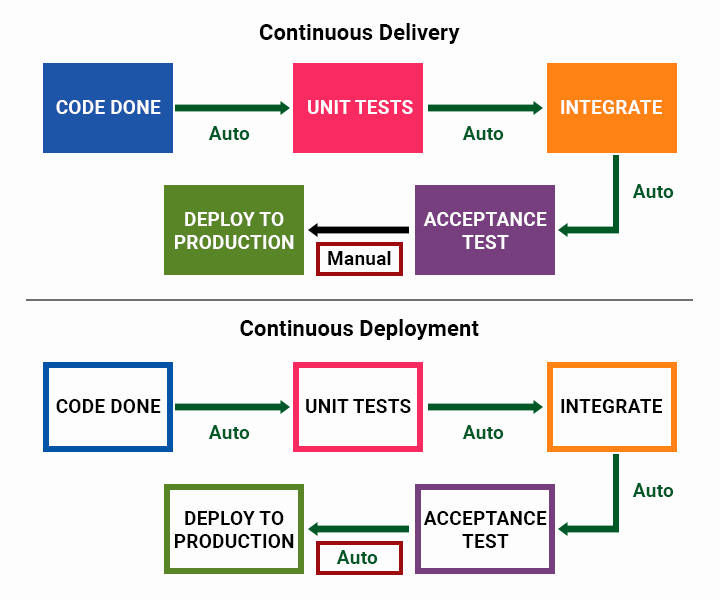
* ***Improve Productivity*** of Developers
* ***Address bugs and issues*** at a very early stage
* ***Faster Delivery*** of software updates

##### Continuous Deployment

**Continuous Deployment** is a Continuous Delivery pipeline with no manual gates between initial code commit / check-in and production.

Some organizations have matured to an extent to deliver multiple updates on the same day.

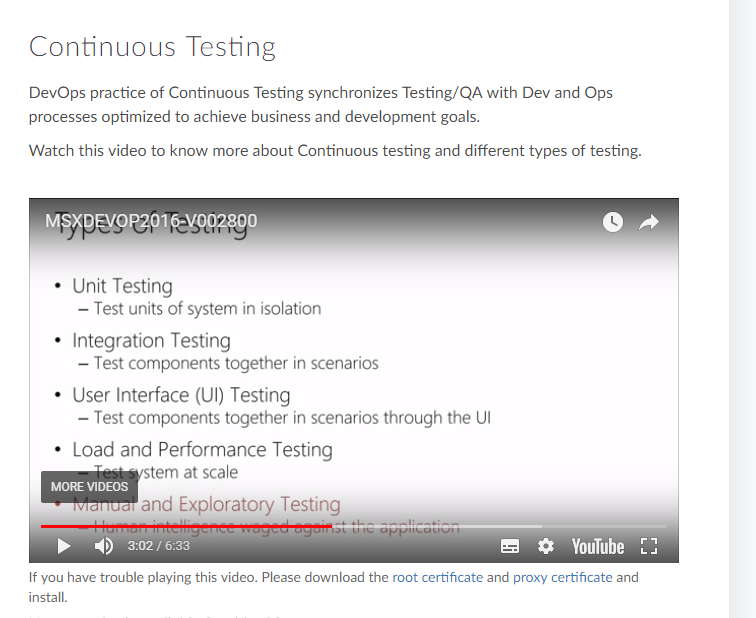
##### Continuous Delivery and Continuous Deployment



##### Continuous Testing

DevOps practice of Continuous Testing synchronizes Testing/QA with Dev and Ops processes optimized to achieve business and development goals.

Watch this video to know more about Continuous testing and different types of testing.



Key Elements of Continuous Testing

* Traceability
* Comprehensive Analysis
* Policy Analysis
* Risk Assessment

Types of Testing

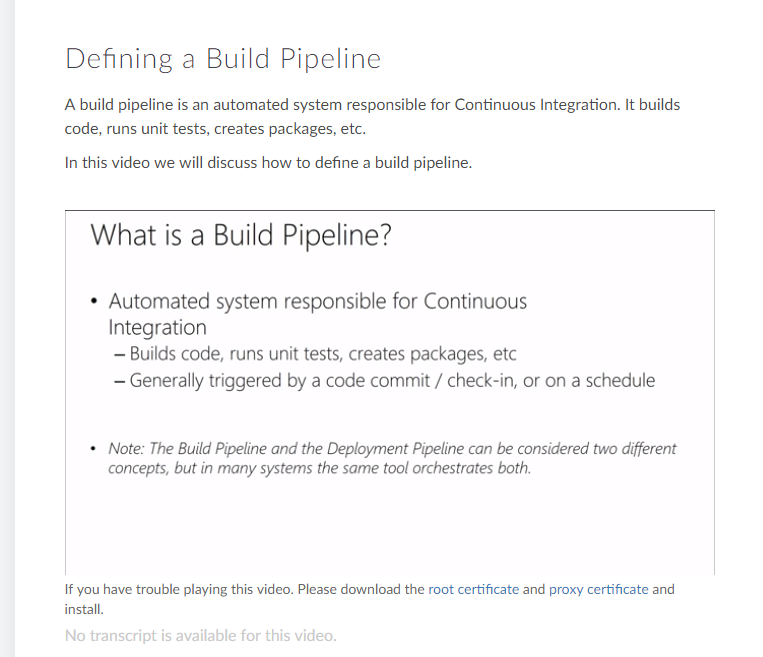
Different Types of Testing that can be automated

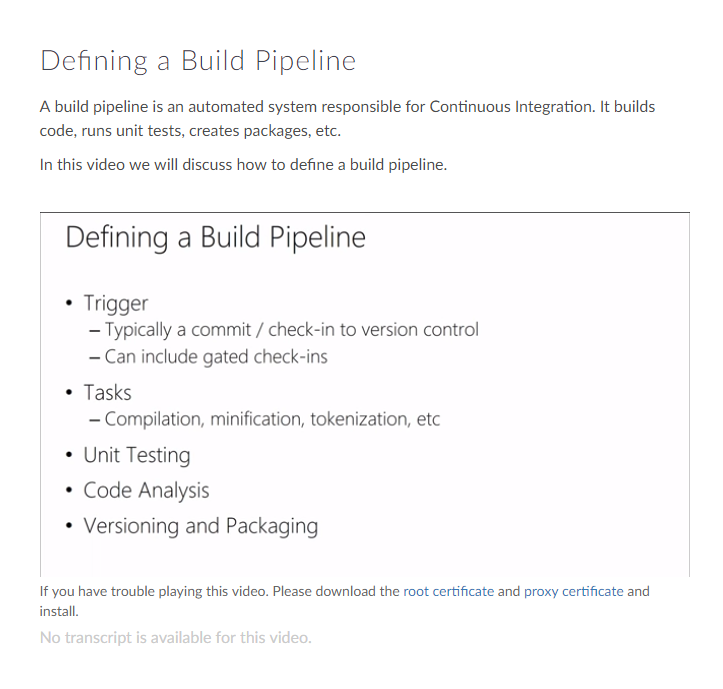
* Unit Testing
* Integration Testing
* Functional Testing
* Smoke Testing
* Regression Testing

##### Defining a Build Pipeline

A build pipeline is an automated system responsible for Continuous Integration. It builds code, runs unit tests, creates packages, etc.

In this video we will discuss how to define a build pipeline.



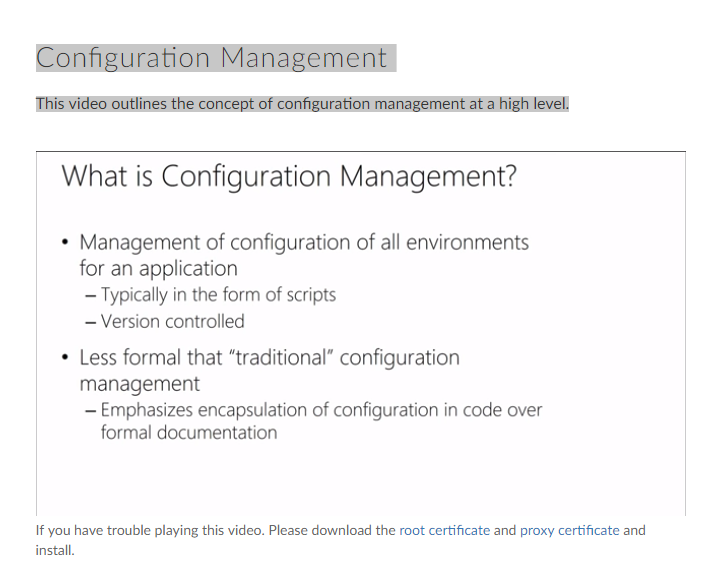


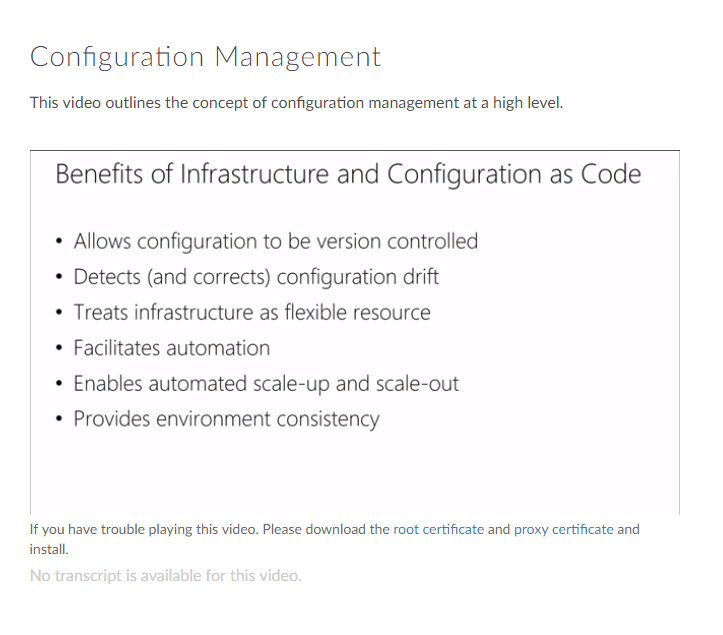
##### Infrastructure as Code

When Infrastructure is treated as code, the knowledge of deployment, configuration management and provisioning are not confined to just System Admins. They can be a developer's tasks.

##### Configuration Management

This video outlines the concept of configuration management at a high level.





Configuration Drift

It is the process where servers running in an infrastructure become different over a period of time due to human induced changes.

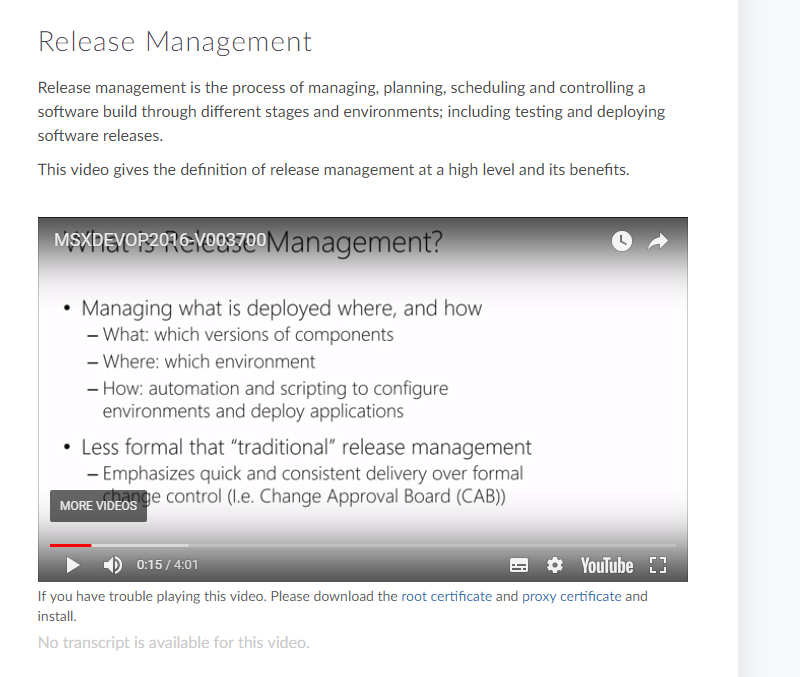
Ways to counter Configuration Drift:

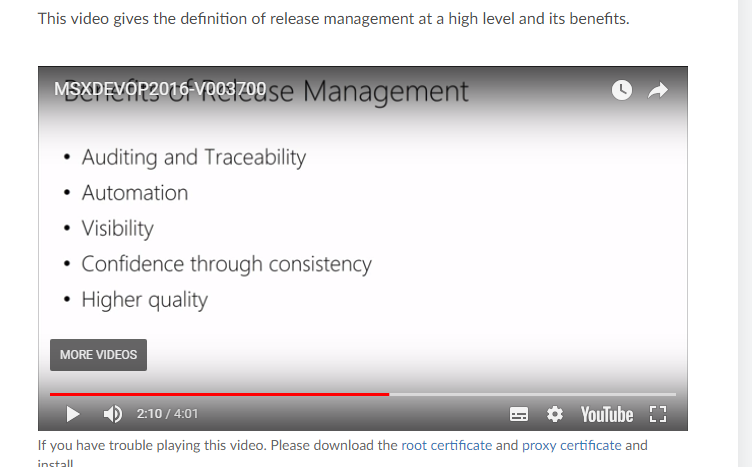
* Rebuilding machine instances frequently
* Make use of automated configuration tools and run them at regular intervals to keep the machines in sync

##### Release Management

Release management is the process of managing, planning, scheduling and controlling a software build through different stages and environments; including testing and deploying software releases.

This video gives the definition of release management at a high level and its benefits.

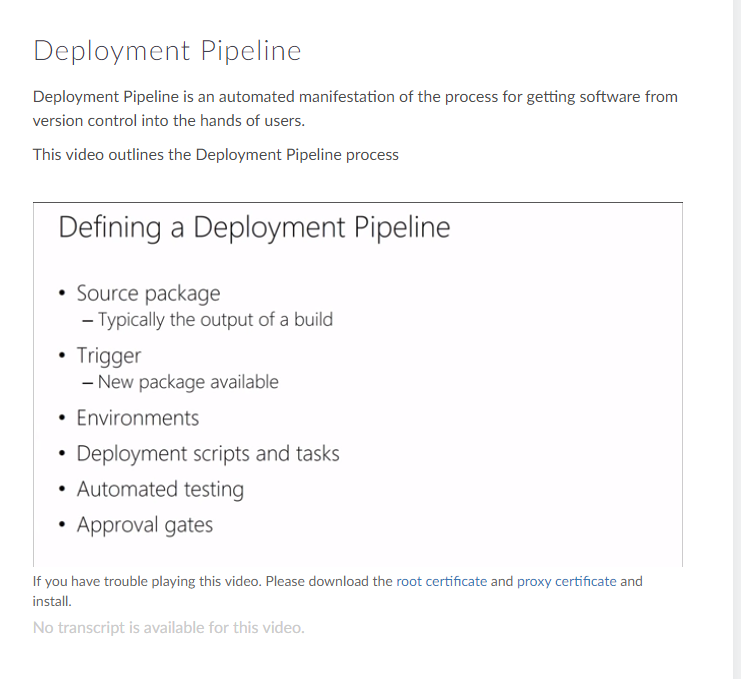




##### Deployment Pipeline

Deployment Pipeline is an automated manifestation of the process for getting software from version control into the hands of users.

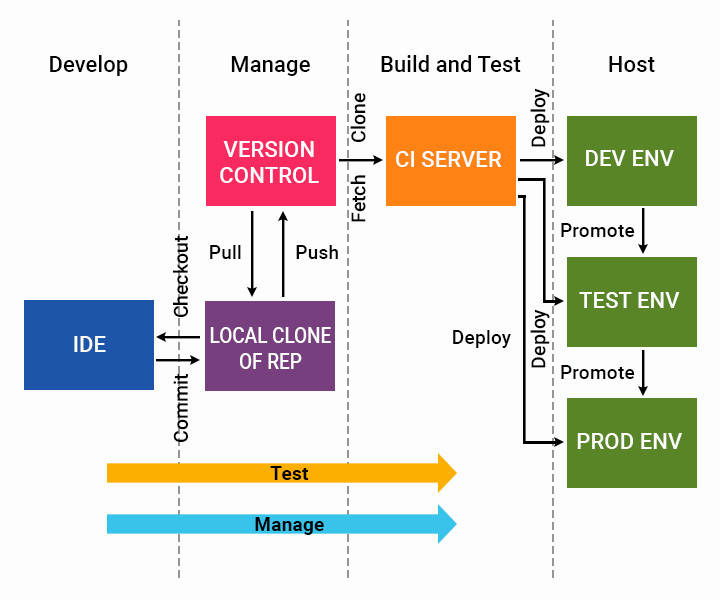
This video outlines the Deployment Pipeline process



Patterns in Deployment Pipeline

* Build Things Once
* Execute Tests in Parallel
* Design Parallel Workflows
* Verify on Environments similar to Production
* Manage Environment in a Pipeline

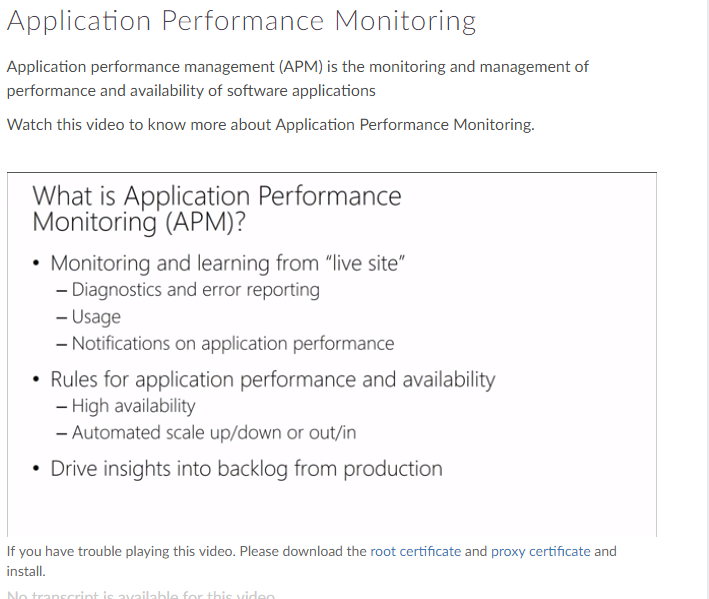
##### Deployment Pipeline Illustrated

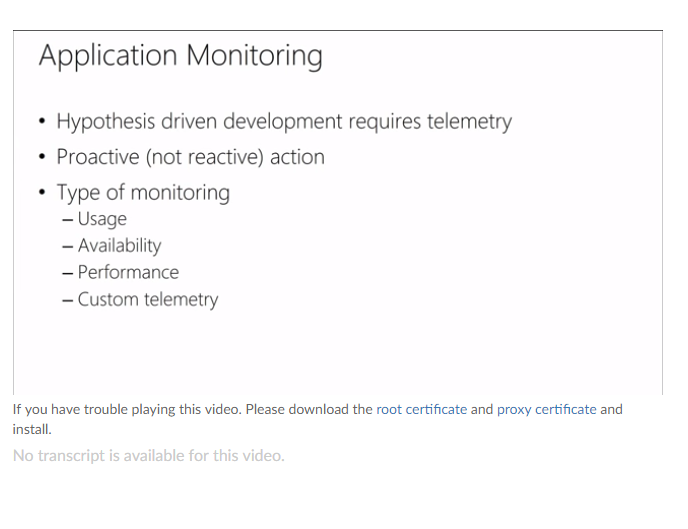


##### Application Performance Monitoring

Application performance management (APM) is the monitoring and management of performance and availability of software applications

Watch this video to know more about Application Performance Monitoring.





Steps to improve Application Monitoring:

* Consolidate Tool Set
* Enrich data with Contextual Information
* Rethink reporting of operational problems
* Automate what doesn't scale

Types of Monitoring

* Usage
* Availability
* Performance
* Customer

Other Metrics in DevOps



Now that you have understood the DevOps lifecycle better, following are some of the metrics that can help you run your team effectively in DevOps.

* ***Cycle Analytics -*** Cycle Analytics measures the time it takes to go from an idea to production for each project you have.
* ***Defect Density -*** Defects detected divided by total lines of code or number of components.
* ***Cycle Time -*** The cycle time is the period required to complete one cycle of an operation, function, or process.

Course Summary

Through this course we have covered the following

* ***DevOps Outline***: Definition and Value of DevOps.How Agile Principles have been used in DevOps.
* ***KPIs and Metrics***: The various metrics used to measure the effectiveness and efficiency of DevOps.
* ***Source Control and Automation*** : The importance of Version Control and Automation in the context of DevOps.
* ***CICD*** : The benefits of Continuous Integration and Continuous Delivery in a DevOps environment
* ***Continuous Testing and Build Pipeline*** : Value Continuous testing can bring to DevOps culture
* ***Infrastructure as Code*** : Benefits of treating Infrastructure as Code , Configuration and Release Management
* ***Application Performance Monitoring*** : The importance of Application Performance Monitoring in the context of DevOps

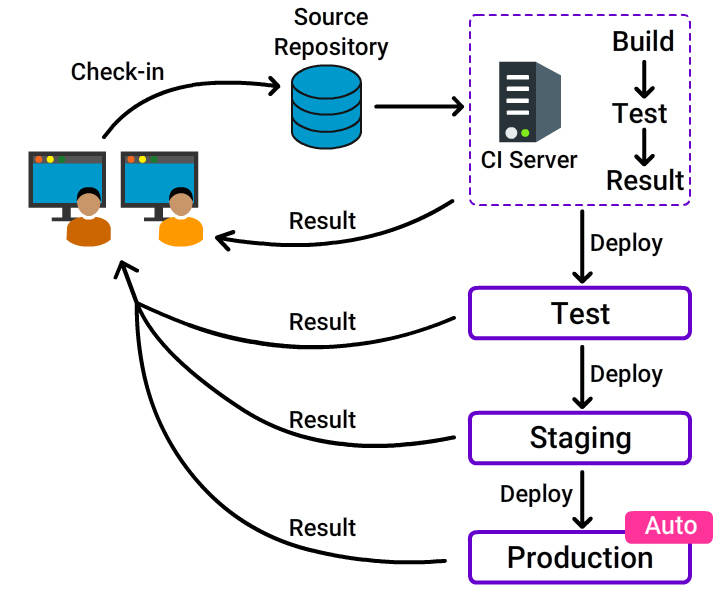
Hope you enjoyed the journey so far.

##### DevOps Lifecycle

##### Plan -> code-> build->test->release🡪deploy🡪operate->monitor

##### Course -2

##### What is Continuous Deployment?



**Continuous Deployment** is a software development practice, where software is built in such a way that it can be released automatically to production any time.

What is Continuous Deployment?...

**CD** is a logical extension of **C**ontinuous **I**ntegration, which **together** aims at giving developers and end users an incredible experience by:

* ***Detecting bugs at an early stage***.
* ***Ensuring seamless deployments*** to all environments.

Why CD?

A reliable release is highly at stake when:

* The software is deployed manually.
* Development is completed and then deployed to a production-like environment.
* Lack of proper configuration tools.

***Adapt*** Continuous Deployment ***for a stable and reliable release***.

##### Why CD?...

* **Reduced cycle time**: Reduces the time it takes from the development of code to the changed code being used by users in production.
* **Zero downtime releases**: Frequent deployment of changes enables negligible production downtime.
* **High quality**: Issues detected early and fixed immediately, ensures the best quality of the end product.
* **Immediate user feedback**: Frequent deployment enables quick user feedback.

##### Continuous Delivery vs Continuous Deployment

We need to **practice *C*ontinuous *D*elivery** in order **to do *C*ontinuous *D*eployment** but the reverse is not mandated.

Foundations of CD

* ***Configuration Management***
* ***Continuous Integration***
* ***Continuous Testing***

##### Configuration Management

**Configuration Management** often used as a **synonym** for **version control**, is the practice where every artifact from source code to build executables, test and deployment scripts, environment configuration details are:

* Stored
* Uniquely identified
* Retrieved
* Modified when needed

Configuration Management...



Configuration management enables:

* ***Reproducibility***
* ***Traceability***

***CM is the backbone*** without which it is impossible to do continuous integration, deployment pipelining and release management.

Environment Configuration

An environment is the:

* ***Set of resources needed for an application to work***.
* ***Configuration of those resources***.

Environment Configuration...

***Attributes*** that ***describe*** an ***environment*** are:

* ***Hardware configuration*** (amount of memory, the number, type of CPUs and so on).
* ***Network infrastructure***
* ***Operating environment***
* ***Middleware configuration*** (software such as database servers, messaging systems, application and web servers).

##### 9th July

##### Configuration Management

**Configuration Management** often used as a **synonym** for **version control**, is the practice where every artifact from source code to build executables, test and deployment scripts, environment configuration details are:

* Stored
* Uniquely identified
* Retrieved
* Modified when needed

Configuration Management...



Configuration management enables:

* ***Reproducibility***
* ***Traceability***

***CM is the backbone*** without which it is impossible to do continuous integration, deployment pipelining and release management.

Environment Configuration

An environment is the:

* ***Set of resources needed for an application to work***.
* ***Configuration of those resources***.

Environment Configuration...

***Attributes*** that ***describe*** an ***environment*** are:

* ***Hardware configuration*** (amount of memory, the number, type of CPUs and so on).
* ***Network infrastructure***
* ***Operating environment***
* ***Middleware configuration*** (software such as database servers, messaging systems, application and web servers).

Integrating Environment Configuration

Environment configuration details can be integrated at:

* ***Build time***: Build scripts pulls the details and integrates with the build deliverables.
* ***Package creation***
* ***Deployment time***: As part of the installation, deployment scripts will fetch and pass the required details to the application.
* Design the application to pull the details during the ***run time or start up***.

Environment Configuration: Practices

* ***Avoid integrating configuration during build or packaging time***. If included at this stage:
  + Same executables cannot be deployed to all environments.

- Tested and released software may not be the same.

* ***Supply configuration information*** for different environments ***using the same source or mechanism***.

- Only one source to be changed, managed and controlled.

- Use registry settings, configuration files, database or an external configuration server like SOAP, REST interface, etc.

Environment Configuration: Practices...

* ***Store configuration options and it's source code in the same repository***.
  + Keep the values (like user details for Github, Twitter etc,.) somewhere else.
* ***Model configuration as a tuple***.

- Tuples vary based on application, it's version and environment it runs.

- Differentiates the configurations used for deploying a particular version of an application.

* ***Test the configuration details***.

- Ensure availability of services the application depends on, like a messaging queue, by smoke testing.

Benefits of CM

* ***Disaster Recovery***: Ability to restore services after a major outage.
* ***Auditability***: Ability to trace back the origin of a deployed change (the version of the code).
* ***Response to defects***: Enables quick fixing of errors.

##### Summary: Configuration Management

* **Use version control**.
* **Store everything in the version control**.
* **Check in changes regularly**.
* **Manage external libraries**.
* **Manage dependencies between components or modules developed by various teams**.
* **Manage application configuration**.

##### Continuous Integration

**The process of automatically triggering build and testing, when a developer checks in a change to the version control is known as Continuous Integration.**

For more details, enroll for the course on Continuous Integration in [***Fresco Play***](https://play.fresco.me/course/169).

##### Testing Philosophy

Cease dependence on mass inspection to achieve quality. Improve the process and build quality into the product in the first place. - Edward Deming

Continuous Testing



***Testing*** is a ***cross-functional activity*** that should be ***done continuously*** from the ***beginning of development***.

Continuous testing:

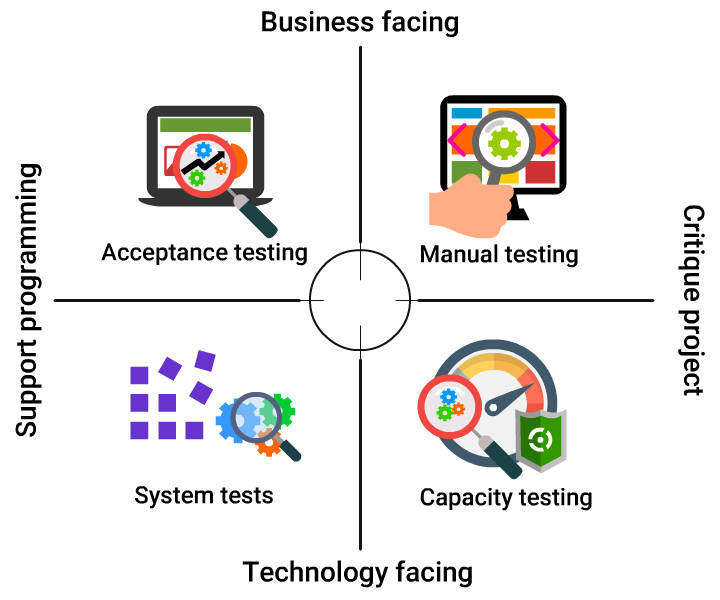
* Provides ***confidence*** that the code is working as desired.
* Results in ***fewer bugs***, improves reputation and reduces support costs.
* ***Encourages good development*** practices.

Testing Strategy

Developers must define strategies for continuous testing, based on:

* ***Risks***: Identify and prioritize the project risks.
* ***Mitigation of Risks***: Decide actions to be taken.

##### Testing Types



User Acceptance Testing



* It is the ***last stage*** of the application's ***testing cycle***.
* This ***test is done using real-time scenarios*** to verify if the specifications are met.
* Also known as ***Operational Acceptance Testing*** (OAT) or ***end-user*** testing.

Types of UAT

* ***Alpha & Beta*** testing: Improves usability of the end product.
  + Alpha: ***Testing*** is done ***in development environment***.
  + Beta: ***Testing*** is done ***in customer environment***.
* ***Contract Acceptance*** testing : Testing is done to ensure ***criteria and specifications***agreed ***in*** the ***contract*** are ***met***.

Types of UAT

* ***Alpha & Beta*** testing: Improves usability of the end product.
  + Alpha: ***Testing*** is done ***in development environment***.
  + Beta: ***Testing*** is done ***in customer environment***.
* ***Contract Acceptance*** testing : Testing is done to ensure ***criteria and specifications***agreed ***in*** the ***contract*** are ***met***.

Types of UAT...

* ***Regulation Acceptance*** testing: ***Tests*** if the ***software complies*** with the ***regulations***.
* ***Operational Acceptance*** testing: ***Tests*** the ***operational readiness*** of the application (such as security, backup, and recovery tests).

Capacity Testing



***Scalability*** testing:

* ***Checks*** the ***response time*** of
  + Individual request.
  + Simultaneous user requests.
* ***More servers***, services etc., ***added*** appropriately ***based on*** the ***test results***.

***Longevity*** testing:

* ***Measures*** the system ***performance***.
* ***Assesses stability*** problems.

Capacity Testing…

***Throughput*** testing:

* ***Checks capability*** of the system ***to handle***:
* - \*Transactions\*.
* - \*Messages or page hits per second\*.

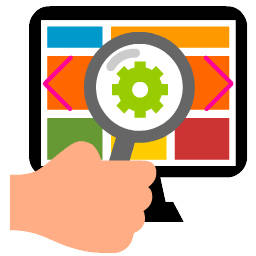
***Load*** testing:

* ***Checks capability*** of application ***to handle an increase in load***.

Capacity Testing Practices

* *Do testing in a production like an environment*.
* *Trigger only if the change passes the acceptance test*.
* *Carefully verify if failures are due to capacity issues (it may fail due to interface issues as well)*.

Manual Testing



**Manual testing** is done to ***ensure*** if the ***application is delivering the value*** as ***expected by the users***.

**Types** of manual testing:

***Showcases***:

* ***Functionality demonstrated*** to users ***frequently*** during development.
* ***Specification issues*** are ***identified*** at an early stage.

Manual Testing...

***Exploratory*** testing:

* ***Explores*** application ***features***.
* ***Leads*** to ***creation*** of a new set of ***test cases***.
* ***Captures new requirements*** for the application.

***Usability*** testing:

* ***Determines*** if the ***application*** is ***useful***, ***usable***, ***accessible*** and ***desirable***.

Smoke Testing



* Originated from hardware testing (a device when switched on is tested for smoke or fire).
* Also known as ***build verification testing***.
* ***Not a comprehensive testing with limited test cases***.
* ***Positive scenarios are tested*** with valid data.
* ***Ensures critical functionalities*** are ***working***.
* ***Identifies issues*** introduced due to components integration.

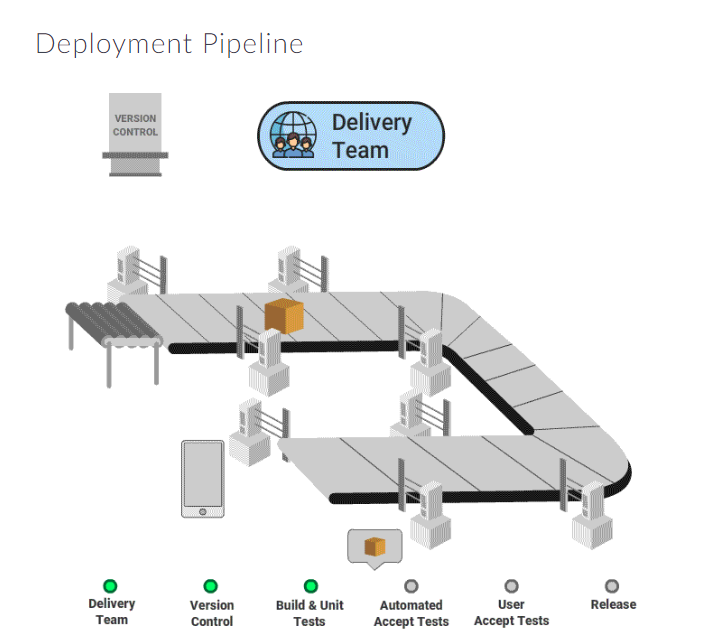
Continuous Testing: Summary

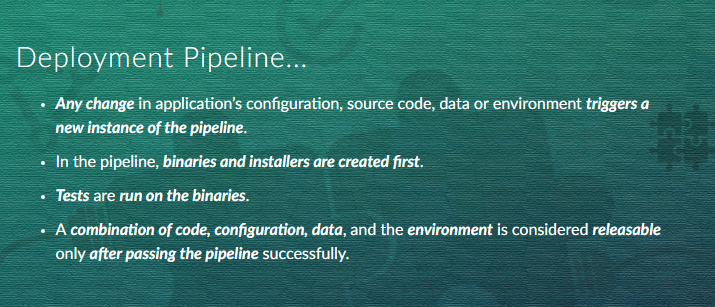
* ***Automate the tests*** (like unit, component, acceptance).
* ***Trigger tests*** when a ***change is done*** to the ***code***, ***configuration*** or the ***environment***.
* ***Perform manual testing continuously*** throughout the project.
* Always ***improve the testing strategy***.

##### Deployment Pipeline

**Automated implementation of an application’s build, test, deploy and release process** is known as ***deployment pipeline***.

Shown are the typical stages a change passes through in the pipeline.

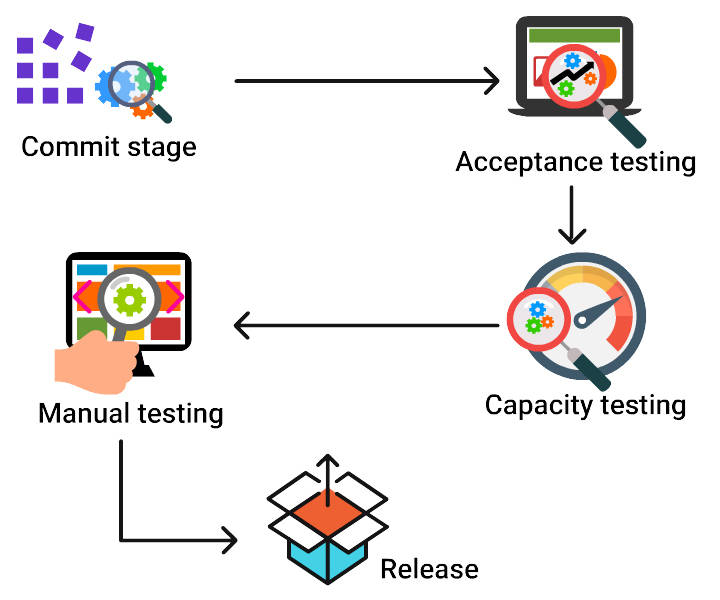




Deployment Pipeline Advantages

* ***Enables collaboration*** (each step is visible to the team).
* Ensures ***issues*** are ***identified*** and ***resolved*** at an ***early*** stage.
* Helps ***team*** to ***automatically deploy and release*** a software version anytime.
* Aids team to ***identify inefficiencies in*** the ***release process***.
* ***Enables metrics collection*** (such as cycle time).

##### Pipeline Flow



Commit Stage

The commit stage includes:

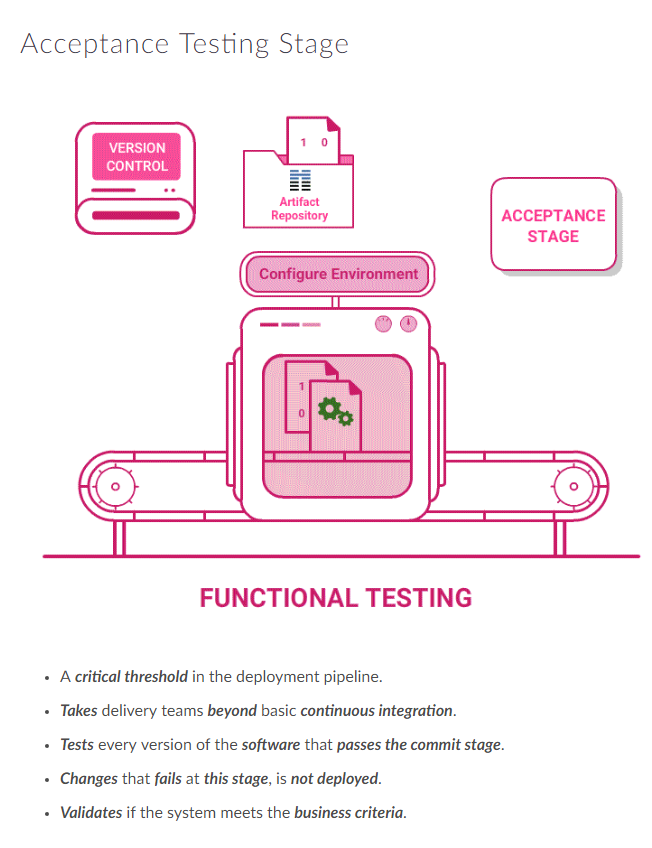
* ***Code compilation***
* ***Execution of commit tests***
* ***Binaries creation*** (for use at later stages)
* ***Static code analysis***
* ***Artifacts creation*** (like test databases)

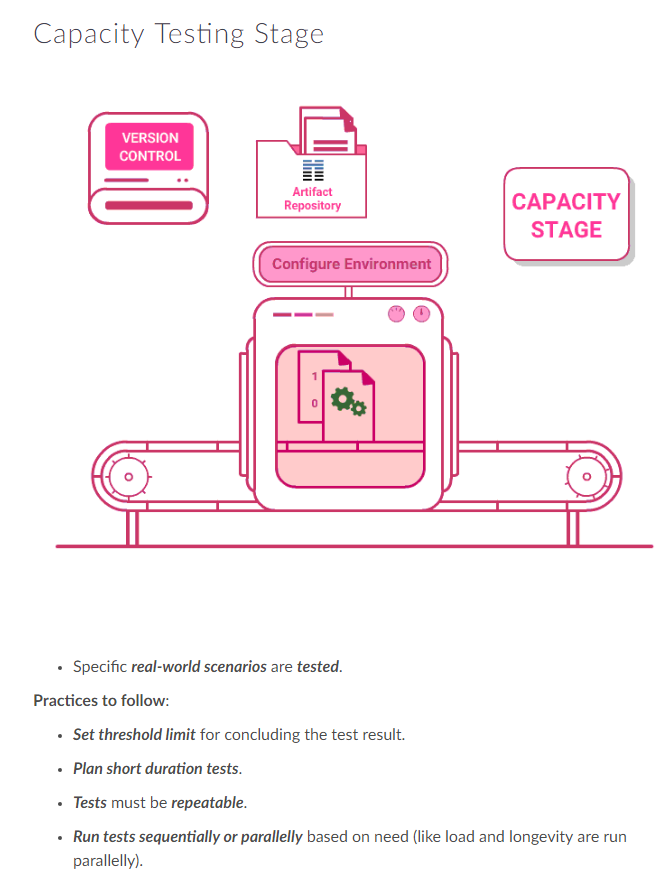
##### Commit Stage...

* The **release candidate** is **created** after passing this stage.
* **Eliminates** the **code changes** that are **unfit** for production.
* **Reports** at the **earliest** to the developer **if the application is broken**.
* The **owner is the developer**.

##### Commit Stage Practices

* **Create efficient, fast, non-environment dependent scripts**.
* **Fail commit**, if the **build fails** due to the compilation, test or environmental issues.
* **Design** a **fast** user **feedback mechanism**.





##### Manual Testing Stage

* **Ensures system** is usable and **fulfills** its **requirements**.
* **Detects bugs** that are not identified in the previous stages.
* This **stage includes**:

- \*\*U\*\*ser \*\*\*A\*\*\*cceptance \*\*\*T\*\*\*esting

- \*\*\*Testing in\*\*\* an \*\*\*exploratory\*\*\* or \*\*\*integration environment\*\*\*.

##### Release Stage

* **Delivers the system** to users:

- \*\*\*As packaged software\*\*\* or

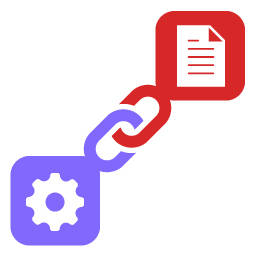
- \*\*\*Deploys it\*\*\* into a production or staging environment.

* Also known as deployment stage.

**Steps to deploy or release software:**

* **Create and manage the infrastructure** (like hardware, networking, etc.) on which your application will run.
* **Install** the correct version of the **application**.
* **Configure the application**, including any data it requires.

##### What are Dependencies?



Dependencies occur, when **a piece of code depends on**:

* another part of the code (**components**) or
* software (like **libraries**) to build or run.

**Dependencies** are **represented** by a **Directed Acyclic Graph**.

##### Dependency Types

* **Build-time**: To be present when the code is compiled and linked.
* **Run-time**: To be present when the application runs in an environment.

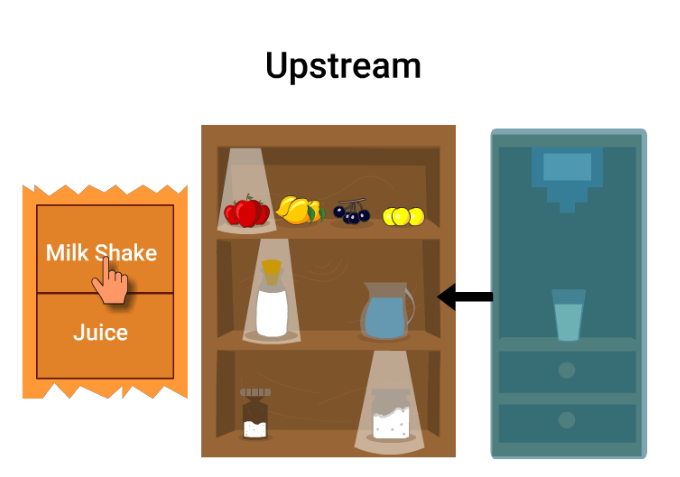
##### Managing Libraries

* **Check-in** the **libraries to** the **artifact repository**.
* **Use a dependency management tool** like Maven or Ivy to declare the libraries. It fetches the libraries from artifact repository when required.

Next section **Component Pipeline** **elaborates** on **components dependency**.

##### Component Dependencies

* **Components towards** the **left** are part of **upstream dependency**.
* **Components towards** the **right** are part of **downstream dependency**.

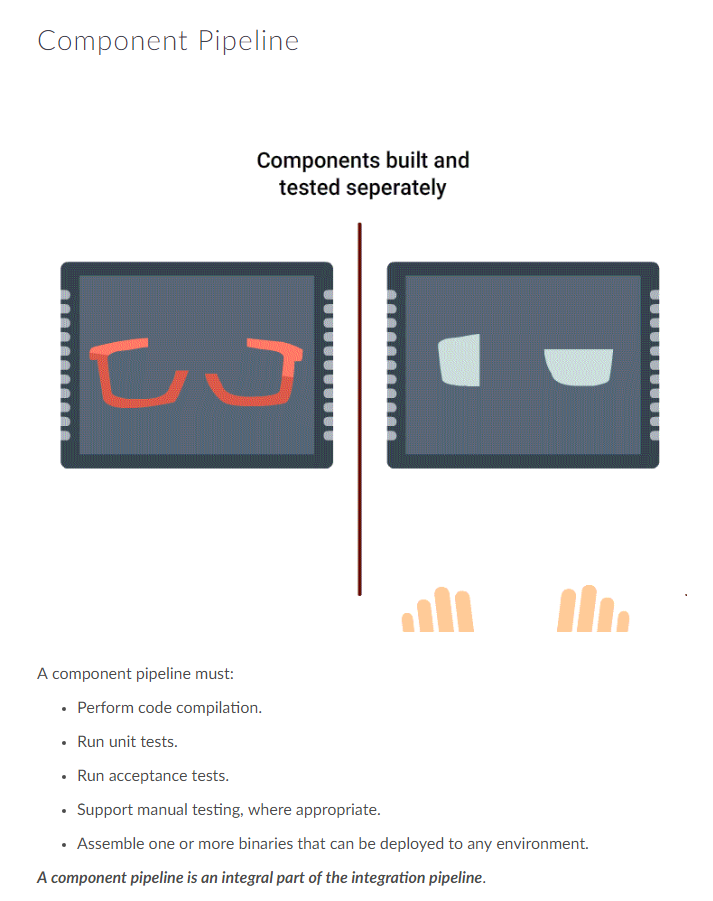


Each component must have:

* Its **own pipeline**.
* **Pipeline triggered** by:
  + **Change in** component's **code**.
  + Any of the **upstream dependency**.



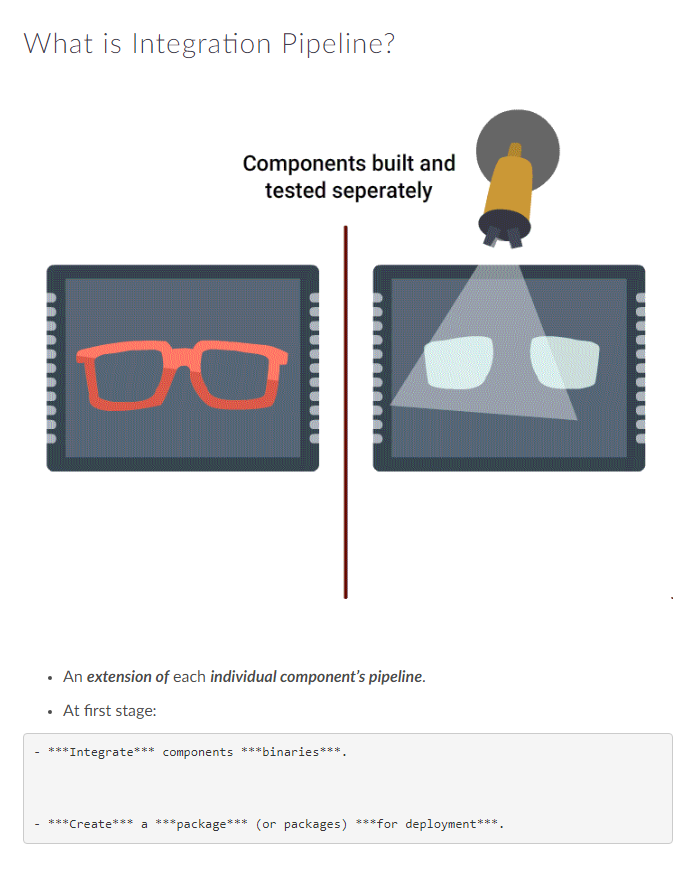




##### Component Pipeline Scenarios

Component pipelines are used in below scenarios:

* **Split application parts with different lifecycle** (like building an own version of an OS kernel and coding can be split, as both changes at a different frequency).
* **Functionally split application areas**.
* **Components using different technologies**.
* **Shared components used by other projects**.
* Stable **components that do not frequently change**.



##### What is Integration Pipeline?...

* At second stage:

- \*\*\*Deploy\*\*\* the application \*\*\*to a production-like environment\*\*\*.

- \*\*\*Run smoke tests\*\*\* to identify any integration issues.

* After **passing** through the **second stage** successfully, **code enters the acceptance test stage**.
* After acceptance test stage, the normal sequence of stages follows.

##### Integration Pipeline Details

* **Components change between** one run of the **pipeline** and the next.
* **Integrating successful builds** of components, **need not necessarily produce** desired **positive result**.
* If the build fails:

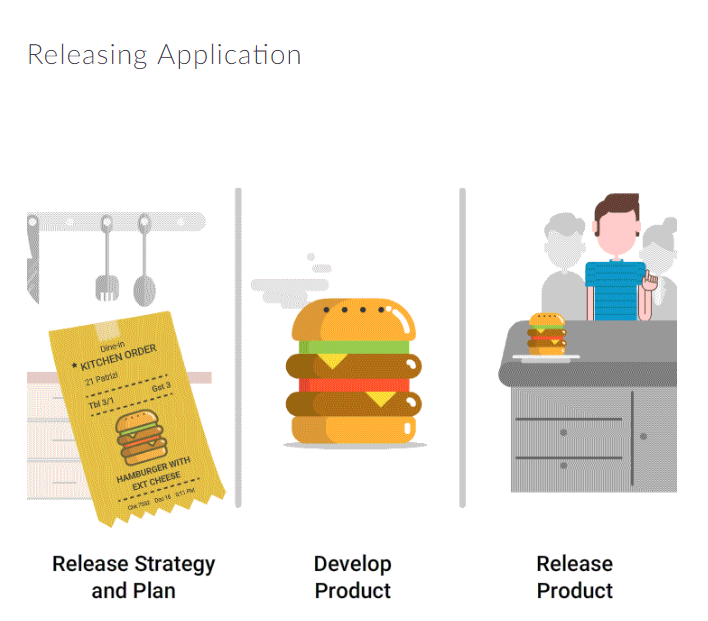
- It will be \*\*\*difficult to conclude the version of the code\*\*\* that \*\*\*caused\*\*\* the \*\*\*failure\*\*\* (code could have changed after the build).

- A simple solution is to \*\*\*build every possible combination of the good versions\*\*\* of the components.

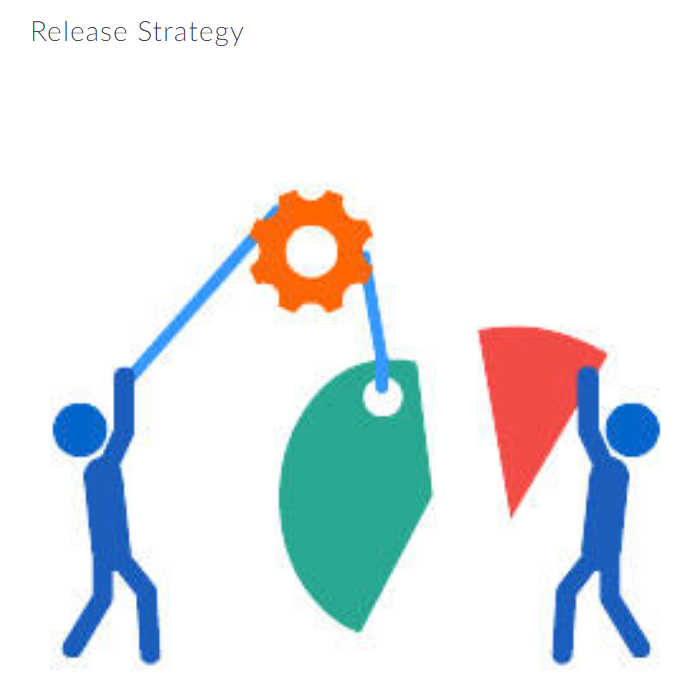
##### Integration Pipeline Practices

Practices to adhere:

* **Provide build status visibility** to the team.
* **Trigger downstream** pipelines, as soon as **binaries pass** through the **unit tests**.
* **Traceability**: Ability to **trace** back the **versions of components** that contribute to a successful or failure build.
* **Plan for short pipelines** (long pipelines or chains of pipelines may compromise feedbacks).



* ***Create*** a release ***strategy***.
* ***Plan*** the first ***release***.
* ***Release*** the ***application***.



Details to be present in the strategy are:

* ***Asset and configuration management strategy***
* ***Technology details***
* ***Deployment pipeline implementation plan***
* ***Test environment details***
* ***Steps to manage configuration***
* ***External*** system ***interface details*** (how and at what stage are they tested).

##### Release Strategy...

* **Disaster recovery plan**
* **Service-level agreements**
* **Production sizing and capacity planning**
* **Archiving strategy**
* **Steps to handle defects**
* **Ways to handle upgrades, including data migration**
* **Application support management**



Release plan ***details*** the ***steps to***:

* ***Deploy*** the ***application*** first time.
* ***Smoke-test*** the ***application*** and its services.
* ***Back out changes***, if the deployment goes wrong.
* ***Backup and restoration*** of the application’s state.

##### Release Plan...

Release plan **details** the **steps to**:

* **Upgrade** without disturbing current state.
* **Restart or redeploy** when the application fails.
* **Monitor** the application.
* **Perform data migrations**.

Release plan contains:

* Information on the **location of the logs**.
* Description of the **information maintained in logs**.
* **Issue log of problems and solutions** from previous deployments.

##### Enabling Application Release



To ensure the application is releasable at any given time:

* **Hide new functionality** until changes are done.
* **Perform incremental changes**, that is releasable.
* **Define components** that change at a different phase.
* **Use branch by abstraction** to make large-scale changes.

##### Implementing Pipeline

To implement a pipeline:

* **Identify** the **value stream** (the processes of a project).
* **Automate build** and **deployment** process.
* **Automate code analysis and tests**.
* **Evolve the pipeline**, based on the metrics collected.

##### Pipeline Practices

* **Build** the **binaries** only **once**.
* **Create** a **script for each stage** in the pipeline.
* **Choose right technology** to deploy the application.
* **Deploy in the same way to all environments**.

##### Pipeline Practices...

* **Deployments must be smoke-tested**.
* **Deploy** into a **production-like environment**.
* Each **change should propagate** to the pipeline **instantly**.
* If **any part of the pipeline fails, stop the line**.
* **Developers and operations must collaborate** on deployment process.
* **Evolve your deployment system** incrementally.

##### Check if Project Uses CD?

Ways to determine if a project is using the concepts of CD:

* At any given time, a particular version of the software can be deployed to any environment (**ready to deploy software**).
* **Automated feedback received** on any change made.

Metrics



Mentioned here are few metrics to track the continuous deployment journey:

***Velocity***: ***Metric to measure the ability*** of the team to complete changes.

***Regression Test Duration***: Regression testing is done to ensure that the changes had not introduced new issues. ***Track the time taken to devise measures to enable quick deployments***.

***Number of open issues***: **Issues that are reported** before a successful deployment.

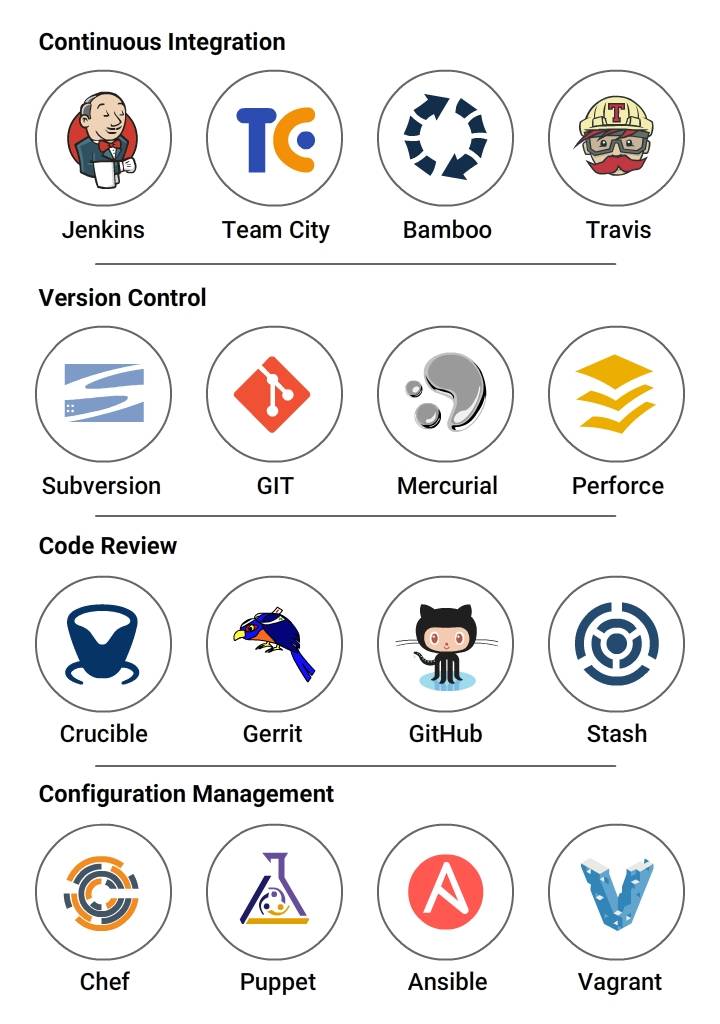
***Percentage of failed deployments***: Measure of **deployments that have failed**.

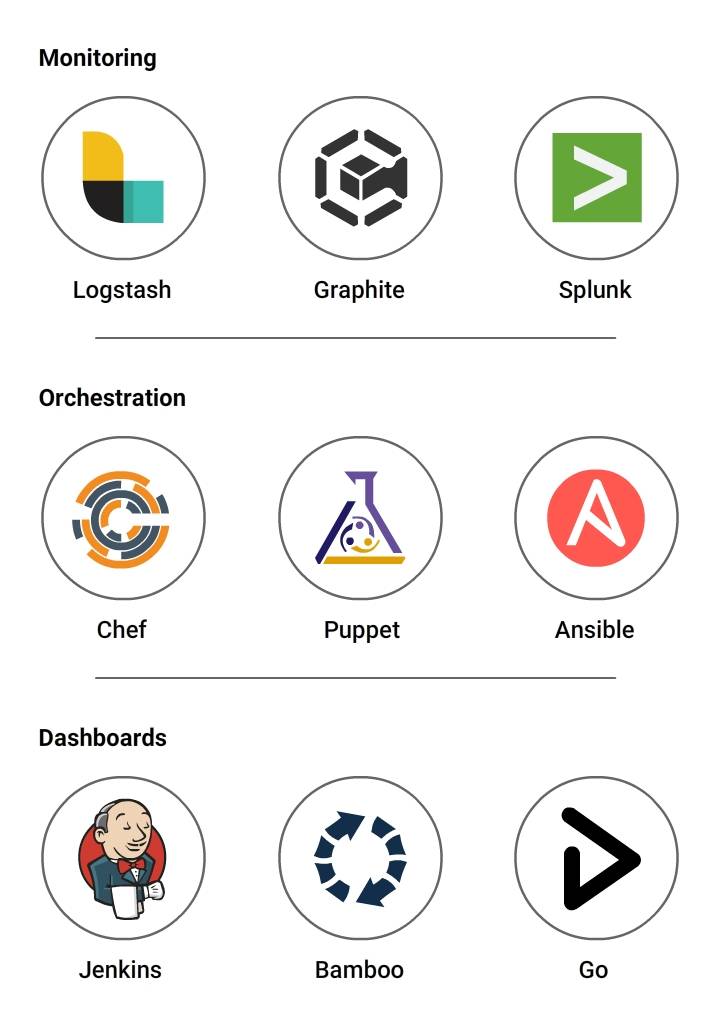
***Broken Build Time***: **Time taken to fix broken builds**.

***Mean Time to Recover***: **Time taken to resolve an issue** after it is reported.

***Cycle Time***: **Time taken between the change and release**.

Tools



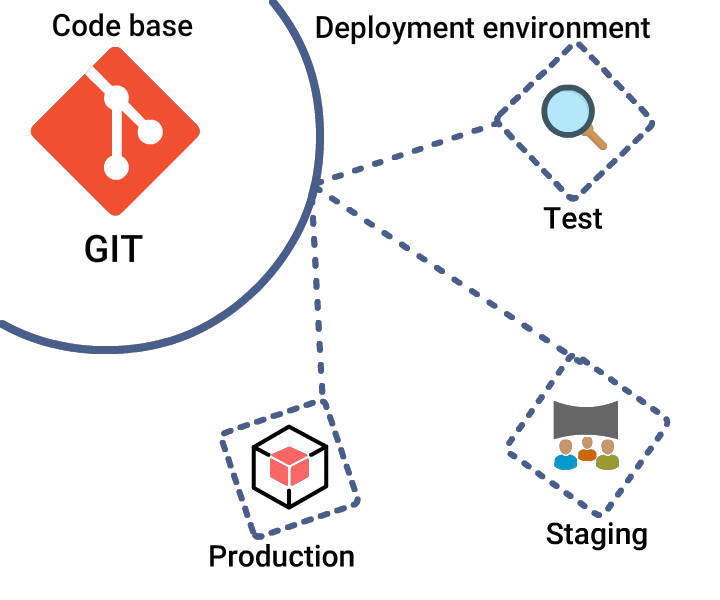


##### CI / CD Rules

Until now, you had read through the concept of continuous deployment.

The main rules to be followed for building and deploying an application are listed in the upcoming cards. (Short summarization of CI and CD concepts).

##### Codebase

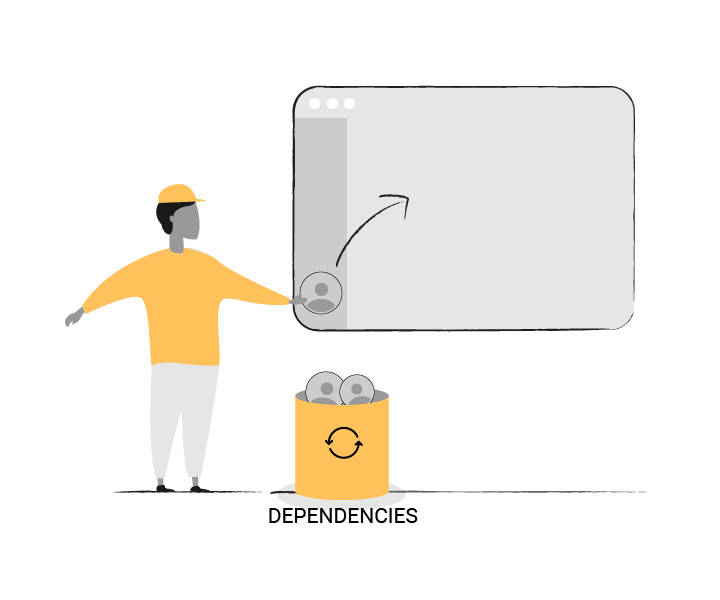


***Rule*: One codebase, many deploys.**

Codebase: Single repository (like subversion) or set of repositories (like Git).

* **One to one correlation must be maintained** between codebase and application.
* **Codebase must remain same across each deployment** (versions can be different).

##### Dependencies



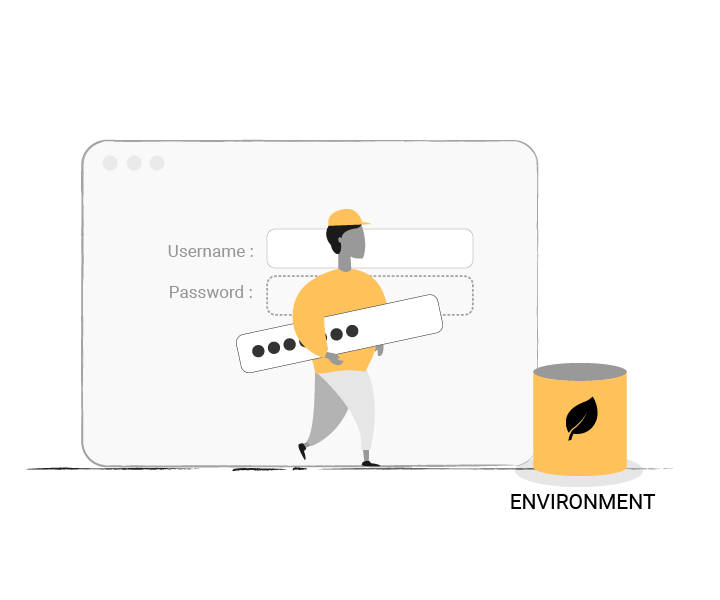
***Rule*: Declare and isolate dependencies.**

* **Maintain a dependency manager**
  + To specify the libraries required for the app.

- Enables a developer to check out the app's codebase.

* **Install language runtime and dependency manager** as prerequisites.

##### Configuration



***Rule*: Store configuration information in environment variables (Env Var).**

Configuration includes:

* **Resource handles** to the database, other backing services.
* **Credentials** for external services like AWS S3.

**Configuration varies across deploys** while code does not.

**Storing** it **in Env Var** makes it **easy to change** without changing code.

##### Backing Services



***Rule*: Treat backing services as attached resources.**

Backing Services are **resources that the application needs for its normal operation** like:

* Data stores (MySQL)
* Messaging systems (RabbitMQ)
* Email services (Postfix)
* Caching systems (Memcached)

##### Backing Services...

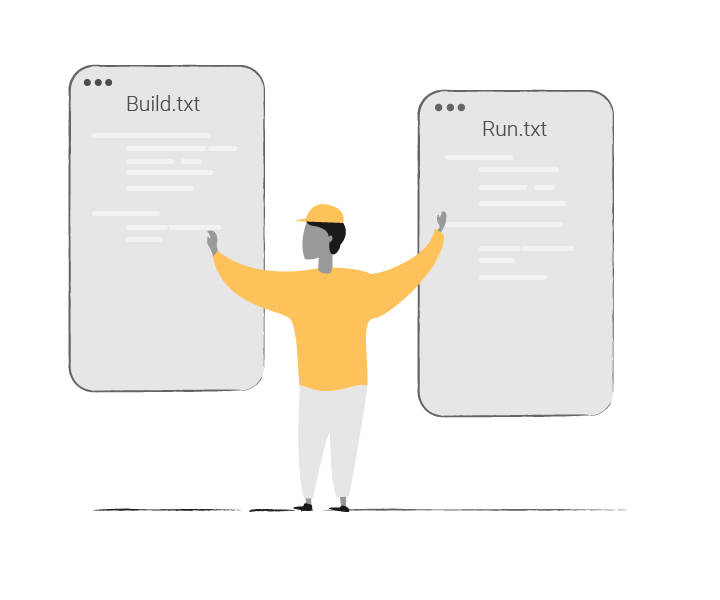
Backing services are **managed**:

* **Locally** by system administrators (like a database).
* **By the third party** (like email services).

Ensure for the app:

* There must be **no distinction between local and third party services**.
* **Services must be easily swapped** without any code change (local with a third party and vice versa).
* **Attach and detach services** when required.

##### Build, Release and Run



***Rule***: **Separate the build, release and run stages.**

Codebase gets transformed into a deploy element in three stages:

**Build stage**:

* Converts a code into an executable bundle.
* Gets dependencies and compiles binaries.

##### Build, Release and Run...

**Release stage**:

* Integrates the build with configuration.
* After this stage, change is ready for immediate execution.

**Run stage (or runtime)**:

* Runs the application.

**Strictly separate build, release and run stages**. For example, do not change a code at the run stage, as it cannot propagate back to the build stage.

##### Port Binding



***Rule*: Export services through port binding.**

At times web apps are executed inside a container. For example, Java apps may run inside Tomcat.

Instead, the app could **export services**:

* **By binding to a port**.
* **Listening to the requests coming in on that port**.

For example:

* Tornado is used for Python.
* Visit a service URL such as http://localhost:5000/ to access the service.

##### Disposability

***Rule*: Fast startup and graceful shutdown.**

* **Start and stop** application **processes when required**.
* **Enables robust** production **deploys** (quick code or configuration deployment).

##### Environment Parity

***Rule*: Keep all environments as similar as possible.**

All environments such as dev, staging and production must use the same type, versions of backing services.

##### Course Summary

To summarize, what you have studied:

* Foundations of CD
* Deployment pipeline and its stages
* Component and Integration pipeline
* Releasing application
* Quick rules to build an app