DOu Certified Tester in DevOps - Foundation Level (DOu CTD-FL) Syllabus

Version 1.2 2021

DevOps United

Copyright Notice

Thank you to the main authors:

- Yaron Tsubery, Vipul Kocher, Umang Agarwal and Sreevatsa Sreerangaraju

Thank you to the review committee

Abdón Sandoval, Aldo Zarza, Alexander Allan, Ángel Rayo Acevedo, Anshu Khandelwal, Arjan Brands, Aurelio Gandarillas Cordero, Bart Knaack, Beatriz López Botello, Christine Green, Claudia Trujillo, Claudio Caravajal Z., Cristián May Jara Segura, Cristian Valdebenito Esponosa, Emilie Potin-Suau, Geoffrey Wemans, Girts Baltaisbrencis, Guino Henostroza, Gustavo Marquez Sosa, Héctor Ruvalcaba, Héctor Santa María Bravo, Javier Chávez, Javier Rojas Cuturrufo, Jean-Luc Cossi, José Antonio Rodriguez, José M. Díaz Delgado, Juan Pablo Rios Alvarez, Julie Gardiner, Julio Córdoba Retana, Karolina Zmitrowicz, Kimmo Hakala, Kyle Alexander Siemens, Laksh Ranganathan, Mani Ananth, Manuel Fischer, Matthias Rasking, Maximiliano Mannise, Michel Dussouchaud, Miguel Angel De León Trejo, Mirco Hering, Miroslav Renda, Norhayati Suzari,

Orane Findley, Paul Mowat, Petr Neugebauer, Radhika Kundur, Ralf Pichler, Richard Seidl, Rik Marselis, Roberto Carlos Galicia Galicia, Samuel Ouko, Sebastian Małyska, Sergio Emanuel Cusmai, Sergio von Borries, Shantel Stewart, Silvia Nane, Sittichai Udomchokpiti, Søren Wassard, Thomas Cagley, Tim Moore, Valeria Cocco, Vanessa Islas Padilla, Vikas Thakur, Wim Decoutere & Wouter Ruigrok.

Revision History

Version	Date	Remarks
DOu A 2019	March 2, 2019	First beta release
DOu B 2019	May 8, 2019	Second beta release
DOu 1.0R 2019	July 16, 2019	First release for review
Version 1.01R 2019	September 3, 2019	Second Review Release
Version 1.03D 2019	January, 2020	Updated Review Release
Version 1.1 2020	April 9th, 2020	First Release
Version 1.2 2021	March 23rd, 2021	Second Release

Table of Contents

Purpose of this document

This syllabus forms the basis of the DevOps United Certified Tester in DevOps - Foundation Level (CTD-FL) certification. This document defines what you need to know in order to pass the certification exam for DOu CTD-FL and is copyright of DevOps United. The certification exam will only cover concepts and knowledge that are described in this document, although this document contains practical elements that will not be covered by the certification exam but is required to be covered in the training.

Resources of the DOu

An overview of DOu resources as well as all relevant information about the DOu certification and other types of DOu certifications are available on www.devops-united.com, the official website of DevOps United. The information to be found on www.devops-united.com includes:

- A complete list of recognized DOu training providers and available courses. Note that training is recommended but not required in order to take the DOu CTD-FL certification exam.
- DOu CTD-FL Syllabus (this document) for download.
- A sample exam set of 10 DOu CTD-FL questions with answers, for training purposes.
- We aim to have the documents available in further languages as soon as possible. For currently available language versions, please check www.devops-united.com.

About DOu Certified Tester in DevOps - Foundation Level (CTD-FL)

DOu Certified Tester in DevOps is a foundation specialist level course for testers involved in DevOps. Our aim at DevOps United is to support you moving one step further into the Agile world, into the DevOps code-based work methodology and culture. You will learn how to use new tools and practices to reduce the traditional distance between programming and systems technicians, in order to build, test, and release software faster and more reliably. This new collaborative approach, DevOps, will allow your teams to work closer together, bringing greater agility to your business and notable increases in your productivity, enabling you to solve critical issues quickly, and better managing unplanned work. This is a practical certification that consists of some theory, and mainly practical, hands-on applications and use of tools, as well as demonstrations, groups and/or individual exercises.

Business Outcomes

BO-1	Understand the business and technology drivers, as well as the methodologies and practices used by DevOps and Continuous Movement in order to create a test strategy
BO-2	Identify and understand the influence of testing on the DevOps movement, how testing is implemented in DevOps and how DevOps fits in various SDLCs
BO-3	Apply automated deployments and automated continuous testing into continuous integration and continuous delivery workflows
BO-4	Apply test types and levels specific to DevOps cycles
BO-5	Apply through hands-on exercises and the interactive practice use of configuration management, continuous: testing, integration, deployment, delivery and monitoring, implementing common DevOps tools such as Dockers, Jenkins, Puppet-Chef/Ansible, Nagios, Cucumber, Selenium, Git/GitHub
BO-6	Understand the basics of Cloud Computing and how it is useful in DevOps
BO-7	Understand the direction and trends related to the future of continuous testing

Learning Objectives/Cognitive Levels of Knowledge

Learning objectives (LOs) are brief statements that describe what you are expected to know after studying each chapter. The LOs are defined based on Bloom's modified taxonomy as follows:

- K1: Remember. Some of the action verbs are Remember, Recall, Choose, Define, Find, Match, Relate, Select
- K2: Understand. Some of the action verbs are Summarize, Generalize, Classify, Compare, Contrast, Demonstrate, Interpret, Rephrase
- K3: Apply. Some of the action verbs are Implement, Execute, Use, Apply

For more details of Bloom's taxonomy, please refer to [BT1] and [BT2] in References.

Hands-on Objectives

Hands-on Objectives (HOs) are brief statements that describe what you are expected to perform or execute to understand the practical aspect of Learning.

The HOs are defined as follows:

- HO-0: Live demo of an exercise or recorded video

- HO-1: Guided exercise. The trainees follow the sequence of steps performed by the trainer
- HO-2: Exercise with hints. Exercise to be solved by the trainee utilizing hints provided by the trainer
- HO-3: Unguided exercises without hints

General Prerequisites

Mandatory

- None

Recommended

- ISTQB® Certified Tester Foundation Level (CTFL) or equivalent
- Basic knowledge of any programming language - Java/Python/R
- Basic knowledge of statistics
- Some software development or testing experience

Programming Language Prerequisites

Required

- Basic knowledge of programming. Understanding of Variables, Functions, Methods, Control Structures (Conditionals and Loops), memory management.
- Basic knowledge of scripting languages.
- Basic knowledge of operating tools such as Selenium, Java, JUnit as well as of basic DevOps tools.
- Basic knowledge of HTTP protocol. Understanding of HTTP Request/Response, and the main elements involved, like Cookies, URL, Parameters, Methods (GET, POST), Headers and Body.
- Basic knowledge of system architecture. Understandings of Web architectures based in layers (Client/Server).

Recommended

- Basic knowledge of SOAP/REST and XML/JSON and/or Web Services or Microservices.

Specific Tools Mentioned in the Syllabus [Disclaimer]

The tools mentioned in the syllabus are used solely as examples. These tools represent some of the most commonly used ones at the time of releasing this syllabus.

This syllabus' focus is on concepts. Therefore, specific tools are used only as the means to demonstrate those concepts, or to perform hands-on exercises. The syllabus does not aim to promote any tool over any other or support any company that produces tools over any other. During training or otherwise, if other suitable alternatives are available, anyone is welcome to use them as well.

Chapter 1 - Introduction to DevOps

Keywords: DevOps, DevSecOps, DevTestOps, DevArchOps, DevWinOps, Pipelines, Continuous Integration (CI), Continuous Deployment, Continuous Delivery (CD), CI/CD, System Center Configuration Manager (SCCM), Shift Left, Software Repository, Code Repository, Branching Strategy, Merging, Test Driven Development (TDD), Behavior Driven Development (BDD), Acceptance Testing Driven Development (ATDD), Specification by Example (SBE).

LO #	Description
LO-1.1.1	Recall the purpose of DevOps (K1)
LO-1.2.1	Explain the components of DevOps (K2)
LO-1.2.2	Recall the core principles of DevOps (K1)
LO-1.3.1	Compare the traditional release process with the delivery pipeline (K2)
LO-1.3.2	Explain the concept of Continuous Integration (CI) and the advantages it offers (K2)
LO-1.3.3	Explain SCCM Concepts: Repositories, Check-in/Check-out, Versioning, Branches, Merging, conflict resolution, working in teams, branching strategies (K2)
LO-1.3.4	Explain CI pipeline and how tools help set up a CI pipeline (K2)
LO-1.4.1	Explain Continuous Delivery (CD) and the advantages it offers (K2)
LO-1.4.2	Recall the tool types used in continuous delivery (K1)
LO-1.5.1	Recall the purpose of continuous deployment (K1)
LO-1.5.2	Compare continuous deployment with continuous delivery (K2)
LO-1.7.1	Recall the main DevOps culture and mindset aspects as well as their importance (K1)
LO-1.7.2	Recall the main reasons why the Shift Left principle contributes to the DevOps (K1)

LO-1.7.3	Recall various DevOps-related terms, such as DevSecOps, DevTestOps, etc. (K1)
LO-1.7.4	Understand how DevOps and Agile fit together (K2)

HO #	Description
HO-1.2.3	Demonstrate the challenges of DevOps (HO-0)
HO-1.3.3	Demonstrate how to apply the main features of a configuration management tool: check-in, check-out, merge, conflict resolution, branching (HO-0)
HO-1.3.4	Create a simple pipeline for code compilation based on trigger from code check-in (HO-1)
HO-1.5.2	Demonstrate how to apply the main features of continuous delivery and deployment tools (HO-0)
HO-1.6.0	Demonstrate the main monitoring elements in tools, such as Nagios or Grafana (HO-0)

1.1 DevOps at a Glance

DevOps is a software engineering culture and practice that aims at unifying software development (Dev) and software operation (Ops). DevOps is a set of practices that combine software development and IT operations to shorten products or the systems development life cycle.

1.1.1 History and purpose of DevOps

LO-1.1.1	Recall the purpose of DevOps (K1)

DevOps initiated after the concepts of fast, multiple, and successful deliveries were presented at the 2009 Velocity conference. The name DevOps was coined to refer to the concept of arranging/creating an infinite cycle between development and operation organizations, in order to enable smooth and incremental product development and delivery cycles.

In 2009, the first conference on the topic, DevOpsDays, was held in Ghent, Belgium. The conference was founded by Belgian consultant, project manager and agile practitioner Patrick Debois.

The main characteristic of the DevOps movement is to strongly advocate automation and monitoring at all steps of software construction, from integration, testing and releasing to deployment and infrastructure management. DevOps aims at shorter development cycles, increased deployment frequency, and more dependable releases, in close alignment with business objectives and customers.

1.2 DevOps concepts

The concept of DevOps is founded on building a culture of collaboration between teams that historically functioned in relative siloes.

1.2.1 Components of DevOps

LO-1.2.1	Explain the components of DevOps (K2)

The key components of DevOps are:

- Automatic delivery of the pipeline
- Configuration management system/solution
- Regular continuous integration
- Automated monitoring and health checks
- Regular continuous delivery
- Infrastructure as a code

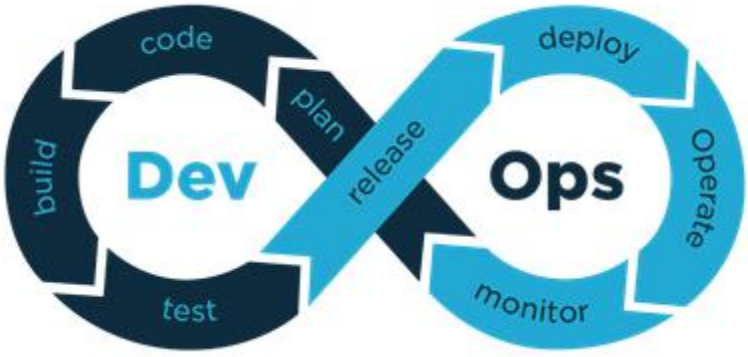
1.2.2 Core principles of DevOps

LO-1.2.2	Recall the core principles of DevOps (K1)

The core principles of DevOps are:

- Customer Centric Action
- Create with the End in mind

- End-to-end responsibility
- Cross-functional autonomous teams
- Continuous feedback
- Continuous integration and deployment
- Continuous improvement
- Continuous experimenting and learning
- Automate everything you can



1.2.3 Challenges of DevOps

HO-1.2.3	Demonstrate the challenges of DevOps (HO-0)

There are many challenges that DevOps is aimed to solve, such as:

- Manual testing
- Test data
- No service virtualization
- Lack of Configuration Management system
- No integrated tools architecture
- Planning in a DevOps environment
- Inconsistent environments
- No production-like environments
- Limited feedback from customers
- Issues in the collaboration between development and operations
- Issues in the collaboration across all product-related departments
- Elicitation of non-functional requirements
- Manual deployments
- Manual releases
- Large releases

- Lack of DevOps metrics
- Lack of traceability across the DevOps landscape

1.3 Continuous Integration

1.3.1 Traditional release process vs. delivery pipeline

LO-1.3.1	Compare the traditional release process with the delivery pipeline (K2)

Traditional release management includes planning, scheduling, monitoring, and controlling a software built at various stages and in different environments. Nevertheless, it may come in many different flavors. Release management guarantees that the production deployments are well orchestrated and follow all the necessary steps to ensure the proper visibility and approvals are obtained throughout the process.
The DevOps, process allows teams to have, or better said, take control over production deployments. The team is, of course, creating working code, but also focusing on the infrastructure, network, and other implementation items necessary to get their code into production. These teams are in charge to create code with higher quality as they take the overall responsibility to have the production systems more reliable and maintainable.
In DevOps, the teams use the data pipeline. It is a set of data processing elements connected in series, where the output of one element is the input of the next one. The elements of a pipeline are often executed in parallel or in the time-sliced fashion. This enables the team to manage it better, utilize its timeline and provide more optimized and controlled delivery cycles.

1.3.2 Definition and principles of Continuous Integration (CI)

LO-1.3.2	Explain the concept of Continuous Integration (CI) and the advantages it offers (K2)

The “continuous” movement consists of the two main approaches, which have merged into what today we call CI/CD. The CI is mainly the practice of merging all the code implemented by the developer(s) and ready to be merged (the work-in-progress code) to a centralized (master) code branch, usually several times a day.

CI stands for Continuous Integration and CD for Continuous Delivery, as well as for Continuous Deployment, which immediately follows.
The CI includes additional processes/mechanisms for making quicker, more reliable deliveries, besides the traditional release process and activities.

The main advantages of the CI pipeline while using smaller deliverable pieces are to:

- Help achieve continuous momentum
- Help catch defects early
- Make defect isolation easier
- Help reduce the costs by automation

The CI/CD are the main enablement rings in the DevOps chain and the main contributors to achieve the DevOps objectives. The long traditional process was broken down by the CI/CD approach into smaller controlled deliverable pieces, to keep the continuous momentum.
The main principle of Continuous Deployment is to enable any new piece of code that passes the related automated tests to be deployed.
The agile movement has set the stage for DevOps, which drives the concept of agile even further, allowing for faster development, while enriching the agile methodology with lots of tools that collaborate and harmonize to achieve a better continuous solution.

1.3.3 Source code configuration management

LO-1.3.3	Explain SCCM concepts: Repositories, Check-in/Check-out, Versioning, Branches, Merging, conflict resolution, working in teams, branching strategies (K2)

HO-1.3.3	Demonstrate how to apply the main features of a configuration management tool: check-in, check-out, merge, conflict resolution, branching (HO-0)

One of the most important and key factors in DevOps is to rely on a robust source code and configuration management (CM) basis.

The SCCM concept or the idea of a central system that manages the configuration management of the code, from its creation till delivery,

production, maintenance, etc. is to have all the necessary steps managed and controlled in one place. The SCCM suite consists of several parts, among which you may find the following key modules and features:

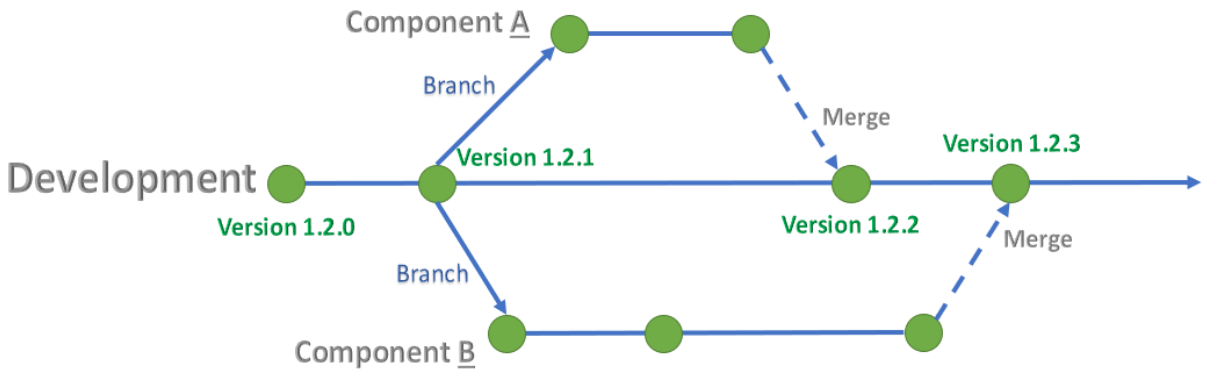
Repositories: The code that developers write is part of an important set of information. Repositories keep and store this information for further use, for restoration, for retrospective aspects, etc. The repositories represent these storages and code holders.

Check-in/Check-out: A programmer writes new code or maintains an existing one. The action of pooling the code from the repository in order to work on it is called “check-out the code”. Any time that the programmer decides that the work is completed and there is a need to store it back as a new or updated piece of code to the main branch repository so that code can be reviewed, the programmer performs an action called “check-in the code”.

Versioning: Versioning is used to label (give a name or number or the combination of name and number) a piece of code that is developed as a program or set of programs — e.g., package. This label represents the name of this specific item or package. The version is a specific reference, which enables the various stakeholders of the products or systems to communicate about them — and allows them to associate the program/product/system with the code that is stored and has the same label. A unique version of the same element (file, document, library, etc.) enables us to have this element available in several versions at the same time and see its involvement.

Branches & branching strategies: A branch is a copy of a piece of source code that allows programmers to develop two versions separately. Branching is a technique utilized to make a copy of the source code in order to create two versions of it, that are developed separately. There are various forms of branching, which can be used to provide a service pack to the customer or a new generation of the product. These kinds of situations require the DevOps team to make a choice, regarding which branching strategy to implement.

Merging: Many branches could possibly be altered by the programmers from the same main branch of code in some cases. For example, the main branch is version 1.2.x, but there are a few development teams that work with this version/branch as a baseline for updating their code, so these are new branches under the main one which is version/branch 1.2.x. This situation, over time, creates complexity, as the code may vary between the new branches that are being created in parallel or from each other. In order to get back to a single place, assembling them all together back in the main branch is required. This activity of reassembling the branches to the main one is named merging.



Conflict resolution: If a failure occurs during the merging process, it indicates a conflict between the current local branch and the branch that is being merged, meaning there is a conflict between the code we wish to merge and another developers' code. The tools available today do their best to merge the code files but leave the conflicts arising the merging process to be resolved manually by the person who operated it.

Working in teams: The programs or systems are usually developed by more than one developer, usually working in a team — in Agile, we may find it in the Scrum team for example. This usually results from the need to have the product or systems ready faster or, where different parts of the system require different programming skills, the development department is divided into teams. Different developers, or even teams may work on the same code — for example, one team may be in charge of the maintenance releases and the other one of new features, etc. The source code and configuration management tools or suites must enable the harmonization of this parallel way of working.

1.3.4 CI Pipeline and tools

LO-1.3.4	Explain CI pipeline and how tools help setup a CI pipeline (K2)

HO-1.3.4	Create a simple pipeline for code compilation based on trigger from code check-in (HO-1)

The CI (Continuous Integration) pipeline is the main backbone in the environment of DevOps. The CI is the first step in the CI/CD in DevOps, then follows the second step which is the CD (Continuous Delivery). Together, they

bridge the gap between the development and the operations sides. A complete CI pipeline consists of three major parts:

- Integrate the code
- Build the code
- Run the unit tests

In order to support bridging this gap, DevOps offers a process consisting of several stages. Each stage is supported by a set of tools that complement each other, and that support the required technologies or tools used in the previous stage.
In order to orchestrate the operation of these stages, the DevOps team uses tools named schedulers (e.g., Jenkins, Github, etc.).

1.4 Continuous Delivery (CD)

1.4.1 CD - Definition and pipeline

LO-1.4.1	Explain Continuous Delivery (CD) and the advantages it offers (K2)

Continuous Delivery (CD) is part of the DevOps method. It consists of stages driven by pipelines, which are designed for businesses that aim to improve applications as well as systems frequently and thus require a reliable delivery process.

The main benefit of CD is that it enables the deployment of code changes to the production in an automated way. This is done by enhancing continuous integration with automated tests and automated deployment once the tests have been cleared.

Continuous integration is enhanced with automated tests and deployment.

An additional advantage of this approach is that it forces the members of a DevOps team to produce software in such a manner that the software can be readily/continuously released whenever required. The entire activity is performed in a sequence of small repetitive cycles. These small repetitive cycle techniques allow the provisioned code to get any type of change incorporated (such as software enhancement, configuration changes, defect fixes, etc.) quickly, in an efficient and sustainable way.

CD requires the automation of the application and system delivery processes with the intent to deploy integrated code into production without critical bugs or delays. The implementation process in the CD stages helps developers merge the new code they have created or changed with the main branch in a consistent way, so they can build an immediate release-ready product.

1.4.2 Tools in CD

LO-1.4.2	Recall the tool types used in continuous delivery (K1)

The growth of DevOps on the market and the need for fostering the CI/CD require that the number of tools that support the CI/CD stages increase and that new code needs to be deployed instantly to the production. These tools are the components of a delivery pipeline that constitutes continuous delivery; they are either serving the CI stages or the CD stages, or both and they have different types and features combinations. They support these stages within the technology and/or domain that the applications or systems consist of, such as web-based, Microsoft-based, Linux-based, Cloud-based applications, etc.

In addition to the above, most are designed to lower the entry threshold to DevOps, by increasing agility, shortening releases timelines, improve the reliability of the software and stay ahead of the competition.

The best-known CD tools on the current market are:

Buddy: is a CI/CD tool for web developers, designed to lower the entry threshold to DevOps. It uses delivery pipelines to build, test and deploy software. In addition, it employs Docker containers [*see more info about containers in 5.1.4 of this syllabus*] with pre-installed languages and frameworks for builds, alongside DevOps, monitoring, and notification actions.

Buildbot: is a software development continuous integration tool which automates the compile or test cycle required to validate changes to the project code base. It is an open-source framework developed in Python.

Urbancode deploy: is a multitier application model or product of IBM. It provides continuous delivery, self-service, rapid feedback, and incremental updates in the agile environment, and automates the application deployments in a consistent manner. Developers can also roll back the applications, organize the changes across servers, tiers, and components.

Codeship: is a powerful tool that automates the development and deployment workflow. Codeship triggers this automated workflow by simply pushing to the

repository. Parallel runs of tests are completed with the **ParallelCI** feature of **CircleCI**.

Strider: is an open-source CI/CD platform. It is written in Node.JS/JavaScript and uses MongoDB. It is published under the BSD license. It supports different plugins that modify the database schema & user interface and register HTTP routes. An extensible framework triggers builds and deployments. It is integrated to many projects like GitHub, BitBucket, Gitlab, etc.

Solano Labs: is also a CI/CD tool that works in the Software as a Service (SaaS) manner of cloud computing. Using Solano, the user can use many languages and frameworks for writing their code and for testing. It can be integrated with other projects, like Github.

Semaphore: supports many languages and frameworks and can be integrated with Github. It performs automatic testing and deployment. Using collaboration, users can invite other collaborators who are all copied from Github.

Wercker: is a tool which automates builds and deploys the container. It creates a unique automated pipeline (build and deploy pipelines) that are executed through the command line interface. It provides the micro-services, which means it triggers the pipelines whenever any new code is committed. Wercker's Docker Stack performs processing very fast and avoids any threat or error. It isolates the applications and services from the operating system.

1.5 Continuous Deployment

1.5.1 Continuous Deployment — definition

LO-1.5.1	Recall the purpose of continuous deployment (K1)

Continuous deployment is a strategy for software applications or systems releases wherein any code commit that passes the automated testing phase is automatically released into the production environment, making changes that are visible to the software's users.

1.5.2 Continuous Deployment vs Continuous Delivery

LO-1.5.2	Compare continuous deployment with continuous delivery (K2)

HO-1.5.2	Demonstrate how to apply the main features of continuous delivery and deployment tools (HO-0)

Continuous deployment means being ready and able to continually deploy, whereas continuous delivery is a state of being ready and able to release any version at any time on any platform.

Both exist in the agile process that provides a framework where there are small, frequent changes, and where feedback is obtained quickly.

In some organizations, DevOps practices such as CI/CD are implemented so that CD refers to both continuous delivery and continuous deployment.

1.6 Continuous Monitoring

HO-1.6.0	HO-1.6 Demonstrate the main monitoring elements in tools, such as Nagios or Grafana (HO-0)

Continuous monitoring in DevOps refers to both the process and the technology which are required to include monitoring across each phase of the company's DevOps and IT operations lifecycles. This method helps to continuously ensure the health, performance, and reliability of the applications and/or systems and their infrastructure as it moves from development to production.

A reliable continuous monitoring is enabled and simplified through the utilization of best practice tools. These best practice tools are the tools that are most relevant at the time the user considers using them (at the time of publishing this document, tools such as Nagios, Grafana and Raygun).

1.7 DevOps in various development practices

1.7.1 DevOps Culture

LO-1.7.1	Recall the main DevOps culture and mindset aspects as well as their importance (K1)

Most companies are organized in three distinct teams: development, testing and operations. Each has their own interests, goals, and methodologies. This

separation often results in miscommunication, delays, and tension. The main characteristic of the DevOps culture is that it improves the collaboration between the R&D (which include development & testing) and operations roles.

The main cultural aspects and mindset of DevOps are the increased collaboration between the departments, a lesser tendency to work in silos, lean work while sharing responsibility, emphasizing teams' autonomy, improving quality, embracing failure while valuing feedback and increasing automation. Many of the DevOps values are agile values, as DevOps is an extension of agile.

1.7.2 DevOps and Shift Left

LO-1.7.2	Recall the main reasons why the Shift Left principle contributes to the DevOps (K1)

The term Shift Left refers to a practice in software development in which teams focus on the quality of their products, as well as work on problem prevention instead of detection, and begin testing as early as possible. An important aspect in shifting left is static testing, in which it is required to review all sorts of inputs for the DevOps teams such as user stories, interface descriptions, and other defining and designing documents, before any code can be created, in order to prevent that faults are being copied and thus wrong code is built. Besides, it adds more static code analysis tools to assist testing the code earlier. Shifting Left also requires the following key DevOps practices: continuous testing and continuous deployment. There are several methods and approaches in software development which are supporting this stage, such as Test-Driven Development (TDD), Acceptance Testing Driven Development (ATDD), Behavior Driven Development (BDD), Specification by Example (SBE), etc. This means that developers join the testing cycles as early as possible in order to prevent problems that, in the past, were detected later and caused delays. The earlier we test, the earlier we detect issues and problems. The above continuous activities methods assist in reducing costs showing up at late testing phases and help provide faster feedback on a change.

The key principles for Shift Left are:

- Test early
- Earlier feedback on customer-oriented use cases
- Test often

- Test incrementally way
- Reduce the cost

1.7.3 DevSecOps, DevTestOps, DevDataOps, etc.

LO-1.7.3	Recall various DevOps-related terms, such as DevSecOps, DevTestOps, etc. (K1)

DevOps evolved over the years and more requirement engineering and process related stages have been incorporated into its frame and have been coined DevOps manner, such as: DevSecOps (for security), DevTestOps (for testing), DevDataOps (for data), DevArchOps (for architecture) and DevWinOps (for MS-centric).

1.7.4 DevOps and Agile

LO-1.7.4	Understand how DevOps and Agile fit together (K2)

“Agile is the ability to create and respond to change. It is a way of dealing with, and ultimately succeeding in, an uncertain and turbulent environment” (Agile Alliance).

In agile, teams are being requested to move faster — introducing enhancements and corrections rapidly and faster — as well as reducing the length of time leading to delivery, while continuing to improve the quality of each release.

The DevOps values and culture are agile-based, and each DevOps stage better defines the required action that the teams need to take as well as the set of tools or suites that support these stages and enable the agile teams — among them the DevOps team — to work faster, in a more decisive and reliable manner. The DevOps approach is serving both process-wise and technically-wise the agility that agile is preaching.

Both agile and DevOps share a common goal, which is to improve the speed and quality of value delivery. The difference is that agile aims to optimize software development specifically but does not take into consideration the other parts of the value stream/chain that come after the development cycles — DevOps definitely does.

Chapter 2 - Continuous Testing

Keywords: Continuous Testing, Testing Quadrant, Static Analysis, Code Coverage, Memory Leak, Performance Measurement, Test Driven Development (TDD), Behavior Driven Development (BDD), Acceptance Testing Driven Development (ATDD), CI Pipeline, DevOps Pipeline, System Under Test (SUT)

LO-2.1.1	Recall the main characteristics of continuous testing (K1)
LO-2.1.2	Understand the modifications to the testing quadrant for DevOps (K2)
LO-2.2.1	Explain Test Driven Development (TDD) and its advantages (K2)
LO-2.2.2	Apply a xUnit framework for performing TDD (K3)
LO-2.3.1	Explain the main features of a static analysis tool (K2)
LO-2.4.1	Explain the main features of a code coverage tool (K2)
LO-2.4.2	Explain the concept of memory leaks detection (K2)
LO-2.4.3	Explain the main tasks of a code performance measurement tool (K2)
LO-2.5.1	Integrate API tests in a CI pipeline (K3)
LO-2.5.2	Integrate a GUI automation test tool in a CI pipeline (K3)
LO-2.6.1	Integrate a behavior-driven development tool in DevOps pipeline (K3)

HO #	Description
HO-2.2.2	Perform a guided exercise of creating tests using the xUnit framework against a given code (HO-1)
HO-2.3.1	Demonstrate how to integrate a static analysis tool into a DevOps pipeline (HO-0)

HO-2.4.1	Perform an exercise providing a hint of integrating a static analysis tool in the DevOps pipeline and checking the test coverage of a given code and its tests (HO-2)
HO-2.4.2	Demonstrate how to Integrate a memory leak detection tool in DevOps pipeline (HO-0)
HO-2.4.3	Demonstrate how to Integrate a code performance measurement tool into a DevOps pipeline (HO-0)
HO-2.6.1	Demonstrate how to integrate a behavior-driven development tool into the DevOps pipeline (HO-0)

2.1 Introduction to Continuous Testing

2.1.1 Definition and characteristics of Continuous Testing

LO-2.1.1	Recall the main characteristics of continuous testing (K1)

Continuous testing is the process of executing automated tests as part of the software delivery pipeline to obtain immediate feedback on the business risks associated with a software release candidate. The main objectives of continuous testing are to test as early and as often as possible. Understanding the variety of tests that need to be covered, considering the main objectives of the continuous testing, and having the right understanding of how to implement them together in the project, help better define the relevant test strategy for this project.
The main characteristics of continuous testing require the following:

- Independent micro (atomic) test cases
- Independence between the tests — to reduce dependencies as much as possible
- Fully automated test environment — having that part of the System Under Test (SUT) as well as testing solutions
- Strong configuration management system to support the evolution of the tests and the automation test solutions to be aligned with the development of the SUT
- Ability to trigger any kind of tests or test types or test levels (different level or mix of levels) — which may also be found in the testing pyramid — against a different environment, different release, or version, etc.

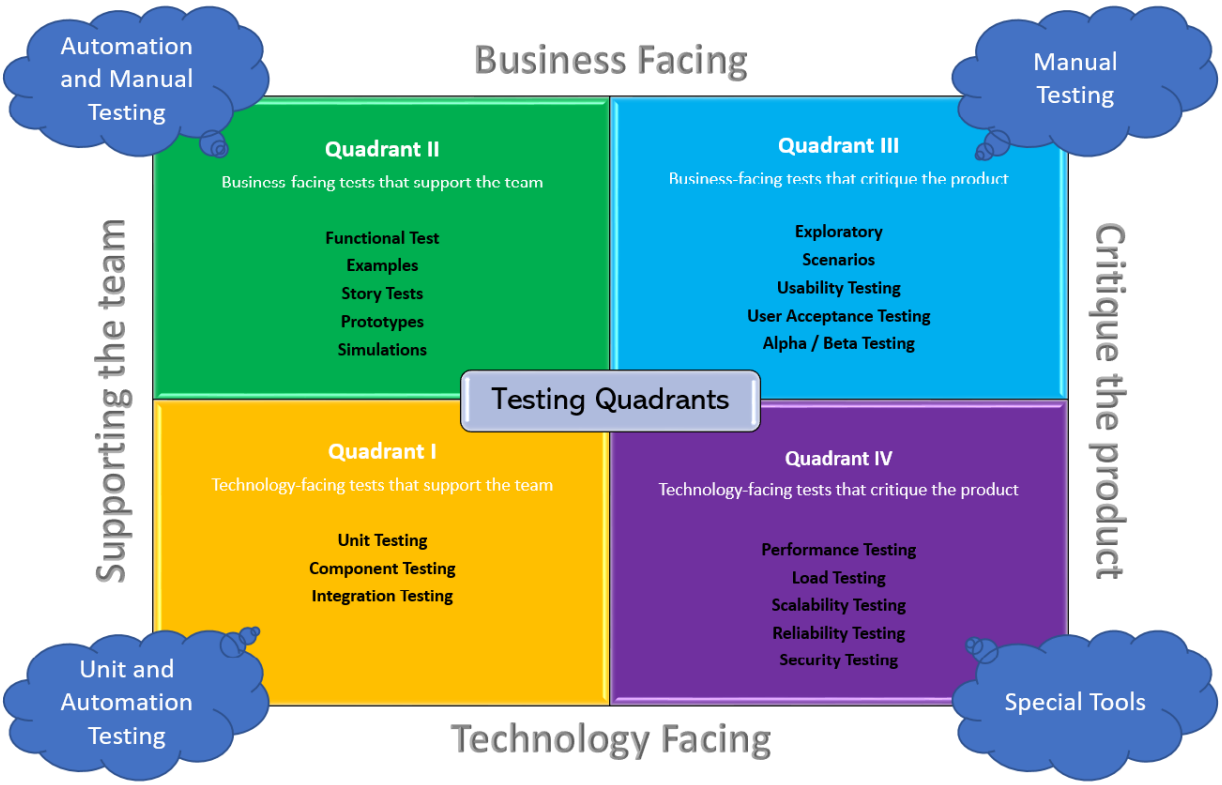
- Understanding that testing is a required activity and not just a phase in the development life cycle.

2.1.2 Testing Quadrant for DevOps

LO-2.1.2	Understand the modifications to the testing quadrant for DevOps (K2)

The Testing Quadrants driven by agile provide a taxonomy in which teams identify, plan, and execute the needed tests. These testing quadrants are both business (user perspective) and technology (developer perspective) facing, they are either manual or automated or the combination of both.

The four testing quadrants are the following:



Quadrant Q1: These are technology-facing tests that are mainly aimed to support the team and development driven methods, such as Unit tests, API

tests, Web Services testing, and Component Tests that improve product design. Tests in Q1 are often associated with automated testing and continuous integration.

Quadrant Q2: These are business-facing tests that are mainly aimed to support the team and development driven methods as well, such as those used for Functional Testing, Story Tests, Prototypes, and Simulations that make sure the software products are properly aligned with the business. Tests in Q2 are often executed using both automated and manual testing.

Quadrant Q3: These are business-facing tests which are used to evaluate or critique the product; mainly covering tests such as exploratory testing, scenario-based testing, usability testing, user acceptance testing, and alpha/beta testing and can involve product demos designed to get feedback from actual users. Tests in Q3 are often executed using manual testing.

Quadrant Q4: These are technology-facing tests which are used to evaluate or critique the product; covering tests such as performance, load, stress, and scalability tests, security tests, maintainability, memory management, compatibility and interoperability, data migration, infrastructure, and recovery testing. Tests in Q4 are often automated.

The main modification in the DevOps quadrants is to automate as much as possible and have the tools that support automation, so these quadrants work together in a more harmonized way.

2.2 Test Driven Development (TDD) and DevOps

2.2.1 TDD – Definition

LO-2.2.1	Explain Test Driven Development (TDD) and its advantages (K2)

The DevOps approach and method encouraging the TDD (Test-Driven Development) approach mainly in a sense of fail fast, fail cheap, has led to put also emphasis on BDD (Behavior-Driven Development) [*see more info about BDD in 2.6 of this syllabus*], which basically is an agile software development process that encourages more collaboration between developers, QA and non-technical or business-oriented participants.

The Test-Driven Development (TDD) is a technique implemented by the developers whereas as a programmer, you must first write the test(s) that in the end fail(s) before writing a new piece of code. This helps to foster good quality in the product from early stages.

Some of the advantages of TDD are:

- Maintainable and extensible code
- Tests that can serve as documentation
- Well defined public interfaces
- Easier and safer refactoring
- Better quality of the code
- Time reduction
- Cost saving in the long run
- Executable acceptance criteria

2.2.2 xUnit Framework

LO-2.2.2	Apply a xUnit framework for performing TDD (K3)

HO-2.2.2	Perform a guided exercise of creating tests using the xUnit framework against a given code (HO-1)

xUnit is a framework, usually consisting of open-source tools, which are available for the community of programmers who are focused on unit testing. Tools such as xUnit.net for the .NET Framework, or JUnit for the JAVA Framework.

2.3 Static Analysis

2.3.1 Coding guidelines and other static tests

LO-2.3.1	Explain the main features of a static analysis tool (K2)

HO-2.3.1	Demonstrate how to integrate a static analysis tool into DevOps pipeline (HO-0)

Static testing consists of several techniques, such as considering design patterns, performing reviews, operating code analysis tools, implementing guidelines, etc.

Among these techniques, we may find coding guidelines that have been created to help detect errors in the early phases of code creation. These guidelines help reduce the extra cost incurred by the software project, as they are related to the type of code that the programmers are writing. They aim to have efficient and effective code by developing in a unified way in order to reduce the possible waste that may appear at the code changes and maintenance stages. When coding guidelines are maintained properly, the readability and understandability of the code increases, thus it reduces the complexity of the code.

This analysis phase can be automated with static analysis tools. There are many tools on the market covering most of the programming languages. The main features addressed by these tools are:

- Checking for adherence to coding guidelines
- Analyzing the code for problematic constructs or potential problem areas such as pointer conversion, uninitialized variables, dead code, etc.
- Code smells
- Security vulnerabilities checks
- Bad programming practices

2.4 Dynamic Analysis

2.4.1 Code Coverage

LO-2.4.1	Explain the main features of a code coverage tool (K2)

HO-2.4.1	Perform an exercise providing a hint of integrating a dynamic analysis tool in the DevOps pipeline and checking the test coverage of a given code and its tests (HO-2)

Code coverage tools assist the developers while implementing and changing the code. They provide abilities to the developers, in which the main of them are:

- to watch and understand how much of their code has been executed while using the application
- to measure the chosen code coverage type
- to be linked to the source code and the test(s) executed on the code
- to report on the monitored code and indicate which areas of the code, were (not) covered by the executed tests

2.4.2 Memory Leaks

LO-2.4.2	Explain the concept of memory leaks detection (K2)

HO-2.4.2	Demonstrate how to integrate a memory leak detection tool in the DevOps pipeline (HO-0)

In order to detect memory leaks, you need to look at the system's RAM usage while the code is running and check whether the program releases correctly the memory that is no longer needed. When the application allocates fresh memory for an operation and loses the reference to the previous allocation, the system may run out of memory sooner or later. This results in system memory getting depleted, slower system responses and eventually a crash or a hang.

Memory leak detection tools help identify the leaks which can then be fixed by the developers.

2.4.3 Code Performance Measurement

LO-2.4.3	Explain the main tasks of a code performance measurement tool (K2)

HO-2.4.3	Demonstrate how to integrate a code performance measurement tool into a DevOps pipeline (HO-0)

The code that is developed by the programmers requires it to be effective, meaning it needs to do the required functionality, and at the same time be efficient with the provided resources (such as memory, CPU, disk-space, etc.). The performance of systems is one of the key indicators for customers' satisfaction, hence the need for early checks on the performance of the code. The main metrics and measurements set for the code performance are related

to resources consumption, such as memory, CPU disk space, etc. It is also important to consider the response time (or latency) of the transactions which the application or the system provide by monitoring and measuring these transactions and their code parts.

Additional measurements are the error rate in the code, the availability of the application over time, and, in some cases, the way that garbage collectors are utilized.

At the level of the code the execution time of various methods is measured including details like the execution time of loops and the number of times a loop is executed and so on. This enables code optimization and performance tuning.

2.5 Integration & System Tests

2.5.1 Integration & System Test Automation — API Tests

LO-2.5.1	Integrate API tests in a CI pipeline (K3)

The Application Programing Interfaces (APIs) are used to connect the different parts of the application or system. API testing is a type of software testing that involves testing the APIs directly and as part of integration testing to determine if they meet expectations for functionality, reliability, performance, and security. Since APIs do not consist of GUI, API testing is performed at the communication layer and includes sending types of requests and analyzing their responses. APIs are either internally or externally used by the application and system, hence, they will be tested either at the Integration or the System test level. The most efficient way to test the APIs is to have these tests integrated in/ to the CI pipelines.

The main steps to perform API tests in a CI pipeline may include the following:

- Build the backend service implementation
- Deploy the backend service to the integration environment
- Deploy the API to the integration environment
- Trigger the execution of the API tests in the deployed environment (either scheduled in that automatic or ad-hoc sequence)
- Report results

2.5.2 System Test Automation — GUI Tests

LO-2.5.2	Integrate a GUI automation test tool in a CI pipeline (K3)

Graphic User Interface (GUI) testing is done mainly at the System test level. The GUI tests consist of both testing the UI objects as well as the functionality provided while using the UI by the customers or end users. There are many tools which enable testers to perform GUI tests. In DevOps, these tools, their tests, and test content also need to be integrated with the CI pipelines. The integration of these tests depends on the platforms and operating systems used to develop and mainly deploy.

The main steps required to integrate the GUI tests to the CI pipeline may include the following:

- Deploy all the services to the system test environment
- Deploy the GUI tests to the system test environment
- Trigger the execution of the GUI tests in the system test environment (either scheduled in that automatic or ad-hoc sequence)
- Report results

2.6 Acceptance Tests

2.6.1 BDD and ATDD

LO-2.6.1	Integrate a behavior-driven development tool in DevOps pipeline (K3)

HO-2.6.1	Demonstrate how to Integrate a behavior-driven development tool into the DevOps pipeline (HO-0)

Behavior Driven Development (BDD) is a method of developing the features based on their behavior. The behavior is basically explained in terms of examples in a very simple language which can be understood by any business expert (e.g., Gherkin language). The Acceptance Testing Driven Development (ATDD) is a development methodology that is meant to bring the customer's view into the development and testing phases. The entire team, that may consist mainly of business customers, developers and testers, collaborate to

define the acceptance criteria (e.g. of an epic or story) before the implementation begins. These acceptance tests are supported by proper examples and other necessary information.

There are DevOps tools available on the market which support the above and enable the DevOps teams to integrate the related tests to the deployed system test environment as well as to the production environments (for customer's pre-production and production testing stages).

Chapter 3 - DevOps specific tests

Keywords: Stage Rollout, Dark Launch, Feature Toggles

LO-3.2.0	Understand the differences between stage rollout, dark launch, and standard upgrade (K2)
LO-3.3.1	Recall the various types of toggles (K1)
LO-3.3.2	Understand the impact of feature toggles on testing in production (K2)
LO-3.3.4	Understand risks associated with Toggles (K2)

HO #	Description
HO-3.1.0	Demonstrate how user specific feature releases can be made using one of the various methods — Internal users, Canary releases, or A/B testing (HO-0)
HO-3.2.2	Demonstrate how dark launch can be implemented for a given system (HO-0)
HO-3.3.2	Perform a guided exercise of creating several functional tests for feature toggles and watch how they can be executed on a given environment (HO-1)

3.1 User specific Feature Testing

HO-3.1.0	Demonstrate how user specific feature releases can be made using one of the various methods - Internal users, Canary releases, or A/B testing (HO-0)

Feature Testing is a process created to provide an insight on the changes made in a software system to add one or more new features or to make modifications in the already existing features. These features have characteristics that are designed to be useful, intuitive, and effective.

3.1.1 Internal user

Internal user testing opens the feature design process to a wider audience. It encourages more collaborative work among team members, so that people acquire a strong design smartness and thoughtful feedback around usability and design.

3.1.2 Canary Release

Canary release is a technique that is used to reduce the risk of introducing a new software version in production by gradually rolling out the change to a small subgroup of users or servers, before rolling it out to the entire platform/infrastructure and making it available to all of the users.

3.1.3 A/B Testing

A/B testing is a process created in order to compare two versions of a single variable (e.g. web page, email, or other marketing asset), typically by testing a subject's response to variant A against variant B, and determining which of the two variants is more effective.

3.2 Stage Rollout, Dark Launch & Standard Upgrade

LO-3.2.0	Understand the differences between stage rollout, dark launch, and standard upgrade (K2)

3.2.1 Stage Rollout

The feature toggle can be set as a system level (meaning for all the users) or for specific users or groups of users.
This enables efficiency and flexibility in the production testing. This also creates a possibility for the stage rollout approach, meaning, scaling up (or down if needed) with the different users that the system contains.
In addition, the staged rollout can only be used for app updates, but not when publishing an app for the first time.

3.2.2 Dark Launch

HO-3.2.2	Demonstrate how dark launch can be implemented for a given system (HO-0)

The ability to develop microservices in an independent approach (e.g., backend service and frontend service — GUI), creates situations where the functionality and business logic side (e.g., backend) are ready, and can be stabilized till the frontend (e.g., GUI) is ready. This is known as dark launch of functionality, and requires tests at this level, mainly to ensure that this part is stable, running correctly and in a robust way.

3.2.3 Standard Upgrade

Standard upgrade is usually escorted by a procedure, either done manually or automatically, or a combination of both. The standard upgrade may require restarts of machines or services in the system, as well as, occasionally, a full or partial downtime of the system or its parts.

3.3 Toggles

The DevOps specific tests are related to the DevOps approach and to the way the features are developed. A good example of the ability to control the appearance or enablement of a feature is the Feature Toggles method and approach. The feature toggles allow the developer to enable or disable the feature easily, either for testing purposes or for real deployment ones. The feature toggles approach creates flexibility while we test, because we can set these toggles (enable or disable them) to have the feature running or not. The flexibility also exists in a situation where the feature is not fully ready to be deployed, hence, through the toggle we can disable it without rolling back or touching the code. It is important to ensure that the feature has no impact on other features when disabled. The feature toggles can be set as a system level, as mentioned above, at the Stage Rollout.

3.3.1 Types of Toggles

LO-3.3.1	Recall the various types of toggles (K1)

There are several types of toggles, such as: release, permanent, Ops, permission and experiment.

3.3.2 Functional Tests for Toggle States

LO-3.3.2	Understand the impact of feature toggles on testing in production (K2)

HO-3.3.2	Perform a guided exercise to create several functional tests for feature toggles and watch how they can be executed on a given environment (HO-1)

The types of toggles that are mentioned may be tested while toggle is off, on, or in the on-then-off modes. The tests can also combine the different toggles, as well as the different types of users that can exist for each toggle or combination of toggles.
In addition, the feature toggles approach creates flexibility while testing as well as if the feature is not fully ready to be deployed on production. The feature toggle can be set as a system level (meaning for all the users) or for specific users or groups of users.

3.3.3 Non-functional Tests for Toggle States

Important test types besides functionality: security, performance, usability, compatibility, and backward compatibility.

3.3.4 Risks of Using Toggles

LO-3.3.4	Understand risks associated with Toggles (K2)

The toggles bring flexibility, hence more complexity, which leads to possible risks, such as: complicated configuration, too many conditions to control as a result of many types of toggles, toggles combinations and the combinations with the different types of users. To better control these conditions, all the data or changes in data (e.g., configuration, provisioning of users, on/off the feature, etc.) need to be documented and had better be separated, to create as little dependency as possible; in addition, enable clean-up and rollback procedures where possible.

In a testing project, it is sometimes recommended to set up a bug hunting session for users (testers or others) testing the system and hunting for bugs. These individuals can receive recognition and compensation for reporting bugs, especially those pertaining to exploits and vulnerabilities. This can be done internally at the company, which may create more collaborations if done in groups or mixed teams between the different departments, as well as if it is encouraged by the product company to be done by the crowd.

Chapter 4 - Operations in DevOps

Keywords: Monitoring, Monitors, Alerts, Controllability, Observability

LO-4.1.1	Recall the concepts of monitoring production system (K1)
LO-4.1.2	Recall the importance of various types of alerts (K1)
LO-4.1.3	Understand the differences between testing monitors and testing alerts (K2)
LO-4.1.4	Understand testing of logging features on a production server (K2)

HO #	Description
HO-4.1.3	Demonstrate alerts in a system using sets of predefined tests for monitors (HO-0)

4.1 Monitoring Production Systems

Monitoring is the process of maintaining surveillance over the existence and magnitude of state change and data flow in a system.

4.1.1 Monitoring

LO-4.1.1	Recall the concepts of monitoring production system (K1)

Monitoring aims to identify faults and assist in their subsequent elimination. The techniques used in monitoring information systems intersect the fields of real-time processing, statistics, and data analysis. A set of software components used for data collection, their processing, and presentation is called a monitoring system.

4.1.2 Alerting

LO-4.1.2	Recall the importance of various types of alerts (K1)

Alerting is the capability of a monitoring system to detect and notify the operators about meaningful events that denote a grave change of state. The notification is referred to as an alert and is a simple message that may take multiple forms: email, SMS, instant message (IM), or a phone call.

4.1.3 Testing of Monitors and Alerts

LO-4.1.3	Understand the differences between testing monitors and testing alerts (K2)

HO-4.1.3	Demonstrate alerts in a system using sets of predefined tests for monitors (HO-0)

The main usage of the monitoring and alerting capabilities is to obtain information — for example, when issues occur in a system based on error codes, the unavailability of features or services, the faults of critical code areas using traps (e.g., SNMP traps for failure in money-related transactions), performance issues (both machine-based and application-based), security issues, usability issues and information about user behavior understanding.

The monitoring and alerts that we have in the systems have to be tested as well. The test types that can be taken for testing the monitoring may include the following:

- Test configuration steps and options
- Test working of monitor by producing situations to be monitored and alerted
- Testability of Monitors
    - Controllability — Ability to provide data to trigger monitors and alerts
    - Observability — Ability to observe the actions taken
- Test removal of monitors

The following test types can be taken for testing the alerts:

- Testing of rules
- Test modification of rules and alerts
- Test the alerts following and while removal of monitors

4.1.4 Log Testing

LO-4.1.4	Understand testing of logging features on a production server (K2)

In addition to monitoring and alerts, the system generates logs and the log types and/or log levels are defined as well. The log level is usually set up in order to provide minimal information or extensive information, for better analysis and troubleshooting activities, thus the correctness of the data provided in the logs needs to be tested. The logging mechanism may consume resources from the system, hence, it is important to know what the required log level to be deployed is, and to test that it provides the required information for this level, as well as consumes minimal resources from the system (e.g., usually low log level is configured, such as errors only, in order to prevent I/O consumption). In addition, for maintenance purposes, it is important to test the resource consumption and the behavior of the system when the log level is high, and more information is provided through the logs.

There are more tests that need to be performed from the functionality perspective for the logs, such as: correctness of the info in the logs, log size, archival, timestamping, unrestricted data encryption/masking, appropriate classification of events (info, warning, error, etc.). As logs create load on the system, performance benchmark is required as well, in order to measure the load and the way that the system performs under normal and heavy conditions while the different log levels are set. In addition, there could be different types of logs (or log files), such as “all logs” only “error logs”, etc. They also need to be tested and optimized as well as to test the traffic which they generate.

Chapter 5 - DevOps and Cloud

Keywords: Infrastructure as a Service (IaaS), Hardware as a Service (HaaS), Platform as a Service (PaaS), Software as a Service (SaaS), IT farm, application containerization

LO-5.1.0	Recall the main forms of cloud computing (K1)
LO-5.1.3	Recall the main advantages of virtualization in the cloud (K1)
LO-5.1.4	Understand the main advantages of cloud computing (K2)
LO-5.1.5	Recall the main differences between Virtual Machines and Containers (K1)

HO #	Description
HO-5.1.4	Demonstrate how Dockers can be applied on a container in a virtualized environment (HO-0)

5.1 Introduction to DevOps with Cloud

Cloud computing consists of three distinct types of computing services, delivered remotely to clients via the internet.
The common cloud business model is that customers typically pay a monthly or annual service fee to the cloud-based providers, to gain access to systems that deliver software as a service, platforms as a service and infrastructure as a service to subscribers. Customers who subscribe to cloud computing services can harvest a variety of benefits, depending on their business needs at a given point in time.
The cloud solutions are similar to what used to be called “hosting centers”, but with additional integration, programming, and operational abilities.

5.1.1 IAAS, PAAS, SAAS

LO-5.1.0	Recall the main forms of cloud computing (K1)

IaaS (Infrastructure as a Service) allows clients to remotely use IT hardware and resources on a “pay-as-you-go” basis from a business model perspective.

It is also referred to as HaaS (Hardware as a Service). Major IaaS players include companies such as IBM, Google, and Amazon. IaaS employs virtualization, a method of creating and managing infrastructure resources in the cloud. IaaS provides small or medium size companies with a major advantage, since it allows them to gradually expand their IT infrastructure without the need for large capital investments in hardware and peripheral systems. If, in the past, each company would create its own IT farm — even for internal use — then the IaaS comes as an alternative option. It is important to note that there are companies which business will impede to use IaaS (e.g., for security reasons — although most of these platforms support a high level of security).

PaaS (Platform as a Service) provides the customers with the ability to develop and publish customized applications in a hosted environment via the web. It represents a new model for software development that is rapidly increasing in popularity. An example of PaaS is Salesforce.com. PaaS provides a framework for agile software development, testing, deployment, and maintenance in an integrated environment. Like SaaS, the primary benefit of PaaS is a lower use cost. PaaS providers handle platform maintenance and system upgrades, resulting in a more efficient and cost-effective solution mainly for enterprise software development.

SaaS (Software as a Service) provides the customers with the ability to use software applications on a remote basis via an internet web browser. Software as a service is also referred to as “software on demand”.
Customers can access SaaS applications from anywhere via the web as service providers host applications and their associated data at their location. The primary benefit of SaaS is a lower use cost, since subscriber fees require a much smaller investment than what is typically encountered under the traditional model of software delivery. Licensing fees, installation costs, maintenance fees and support fees that are routinely associated with the traditional model of software delivery can be virtually eliminated by subscribing to the SaaS model of software delivery. Examples of SaaS include Google Applications and internet-based email applications such as Yahoo! Mail, Hotmail, and Gmail.

DevOps has a great value in the development of SaaS applications that run on the very infrastructure the cloud provider offers, DevOps also can be utilized to assist with the migration of applications to other cloud computing models such as platform as a service (PaaS) and infrastructure as a service (IaaS).

5.1.2 Fitment of Cloud in DevOps

The main advantages of cloud computing are: 24/7 availability, flexibility related to the capacity of the system, ability to easily scale up and down per the required consumption, automated and rapid updates on the software, enhanced collaboration, reduced maintenance, and the ability to provide a frame to scale up small companies that now can reduce their IT costs.

5.1.3 Virtualization and Cloud Computing

LO-5.1.3	Recall the main advantages of virtualization in the cloud (K1)

The cloud is mainly based on virtualization.

The main advantages of the virtualization in the cloud are:

- Offering the ability to access powerful IT resources on an incremental basis
- Levelling the playing field for small and medium sized organizations
- Providing the necessary tools and technology to compete in the global marketplace
- Avoiding the requisite investment in on premise IT resources

This is also applicable for customers who subscribe to computing services delivered via the “cloud”. They can greatly reduce the IT service expenditures for their organizations, and gain access to more agile and flexible enterprise level computing services, in the process.

5.1.4 Application Containerization

LO-5.1.4	Understand the main advantages of the cloud computing (K2)

HO-5.1.4	Demonstrate how Dockers can be applied on a container in a virtualized environment (HO-0)

The fact that cloud computing is centralized by nature, this provides DevOps automation with, for example, a standard and centralized platform for testing, deployment, and production. Using a cloud platform solves many issues with distributed complexity.

In addition, DevOps automation is becoming more cloud-centric. Most public and private cloud computing providers support DevOps systemically on their platforms, which include continuous integration and continuous development tools. This tight integration lowers the costs associated with on-premises DevOps automation technology and provides centralized governance and control for a sound DevOps process.

Virtualization has been in existence for long and its adoption is increasing day by day, and the utilization resources are becoming more and more developed and necessary. The DevOps approach has shifted even further the enhancement of the virtualization world, and more advanced technologies have grown in these virtual fields, such as Docker containers. Docker containers are designed to run on every environment, from physical computers to virtual machines, from bare-metal, Clouds, etc.

Application containerization is a lightweight alternative to having a virtual machine that includes the encapsulation of an application with its own operating system in one place — a container. The term “container” takes its meaning from logistics terminology, packaging container.

5.1.5 Virtual Machines and Containers

LO-5.1.5	Recall the main differences between Virtual Machines and Containers (K1)

To create a cloud application, we can use microservices as an architectural approach, where each application is built as a set of services. Each service runs its own processes and communicates through application programming interfaces (APIs).

Docker works by providing a standard way to run your code. Docker is an operating system for containers. The way a virtual machine virtualizes (removes the need to directly manage) server hardware, containers virtualize the operating system of a server. Docker is installed on each server and provides simple commands you can use to build, start, or stop containers.

Containers are an abstraction at the app layer that packages code and dependencies together. Multiple containers can run on the same machine and share the OS kernel with other containers, each running as isolated processes in user space.

The current, most popular cloud platforms are: Amazon Web Services (AWS), Microsoft Azure, Google Cloud Platform and Alibaba Cloud.

Chapter 6 - Various Tools and Technologies

Keywords: Infrastructure as code (IaC), binary repository, Research & Development (R&D), software repository

LO-6.1.0	Recall the main advantages of modelling infrastructure with code (K1)
LO-6.1.3	Apply the main features of an infrastructure as a code tool (K3)

HO #	Description
HO-6.1.3	Demonstrate how various IaC tools can be orchestrated and working in a harmonized way (HO-0)

6.1 Infrastructure and Repositories

LO-6.1.0	Recall the main advantages of modelling infrastructure with code (K1)

6.1.1 Infrastructure as Code (IaC)

Infrastructure as code (IaC) is a lightweight and data-driven way of managing and provisioning the system's infrastructure (networks, virtual machines, load balancers, and connection topology) — basically treating your servers, databases, networks, and other infrastructure like software. It is done in a descriptive model, using the same versioning as a DevOps team uses for source

code, rather than managing it with physical hardware configuration or interactive configuration tools.
It has similar principles as a code management: the same source code generates the same binary, and an IaC model generates the same environment every time it is applied — e.g., the release pipeline executes the model to configure target environments.
That way, any time the team members need to make changes, they can edit the source code and not the target environment.
IaC is a key DevOps practice and is used in conjunction with continuous delivery.

The main advantages of having modelized IaC are:

- Faster and simpler
- Avoiding deployment inconsistencies of the system's configuration
- Reducing risks
- Increasing R&D's efficiency and productivity, especially in the cloud
- Cost reduction & saving

6.1.2 Binary Repositories

A binary repository, or software repository, or "repo" for short, is a storage location for software packages. Usually, a table of contents is stored, as well as metadata. Repositories manage group packages as well. Sometimes, the grouping is for a programming language, sometimes for an entire operating system, sometimes the license of the contents is the criteria.

The server side of a software repository is typically managed by source control or repository managers. There are repository managers which allow for the aggregation to other repository locations into one URL and provide a caching proxy. While creating continuous builds and generating a lot of artifacts, the repository managers often store them in a central place, and it is important that these repository managers will also be in charge to automatically delete the builds which are not released.
On the client side, package managers help installing from and updating the repositories.

As part of the development lifecycle, the source code is continuously being built into binary artifacts using continuous integration. This may interact with a binary repository manager, similar to a situation a programmer would face by getting artifacts from the repositories and pushing builds there.

A tight integration with CI servers enables to store important metadata such as:

- Which user triggered this build? (manually? or by committing to version control?)
- Which modules were built?
- Which sources were used? (commit id, revision, branch)
- Which dependencies were used?
- Which environment variables were used?
- Which packages were installed?

6.1.3 IaC Tools

LO-6.1.3	Apply the main features of an infrastructure as a code tool (K3)

HO-6.1.3	Demonstrate how various of IaC tools can be orchestrated and working in a harmonized way (HO-0)

The IaC tools are classified in the following two categories, which often might overlap:

- Configuration orchestration tool: these tools are designed to automate the deployment of our infrastructure.
- Configuration management tools: these tools are designed to help configure the software and systems on the provisioned infrastructure.

Some of the current IaC tools allow developers to orchestrate/provision infrastructure as well as configure it, and since the infrastructure gets more complex, it is common to see both types of tools used.
Here are some of the current tools that are popular in the IaC landscape:

Terraform is an infrastructure-provisioning tool that supports multiple cloud providers. It allows developers to represent their infrastructure, regardless of whether it is AWS, Google Cloud or Azure, in a well-defined language known as HCL. Terraform is highly extensible via plugins and has a very strong community around it that produces great open-source modules for provisioning chunks of infrastructure.

AWS CloudFormation is not cloud agnostic and is dedicated to provisioning infrastructure within AWS accounts. It allows for the representation of the AWS infrastructure in JSON or YAML configuration files that can execute to create, update, and destroy resources. CloudFormation is very well integrated into the AWS ecosystem and has developed a lot of features for previewing, rolling back, and managing resource changes.

Chef falls under the configuration management tooling. It allows for the creation of recipes and cookbooks that define the exact steps needed to reach the necessary configuration of the application. Chef, like Terraform, supports multiple cloud providers. It uses the commonly known language, Ruby. It is typically used for configuring Elastic Compute (EC2) instances and even on-premises servers.

Puppet is another Ruby-based configuration management tool, like Chef. The difference between them is that Puppet is known as a declarative tool. This means that developers define what their infrastructure is supposed to be, and Puppet figures out how to make that possible.

Ansible is an infrastructure automation tool from Red Hat. Developers describe how their components and system relate to one another. It is meant to manage and codify systems end to end rather than independently. These definitions are written in YAML and are known as Playbooks.

Juju is an IaC tool from Ubuntu, which allows developers to represent their infrastructure as charms which are sets of scripts that deploy and operate systems. These charms can be packed together as bundles to deploy the entire infrastructure for an application.

uDeploy is an automation deployment framework that reduces deployment errors and improves efficiency, correctness, and traceability. The IBM **UrbanCode Deploy** is a tool for automating application deployments through our environments. It is designed to facilitate rapid feedback and continuous delivery in agile development while providing the audit trails, versioning and approvals needed in production.

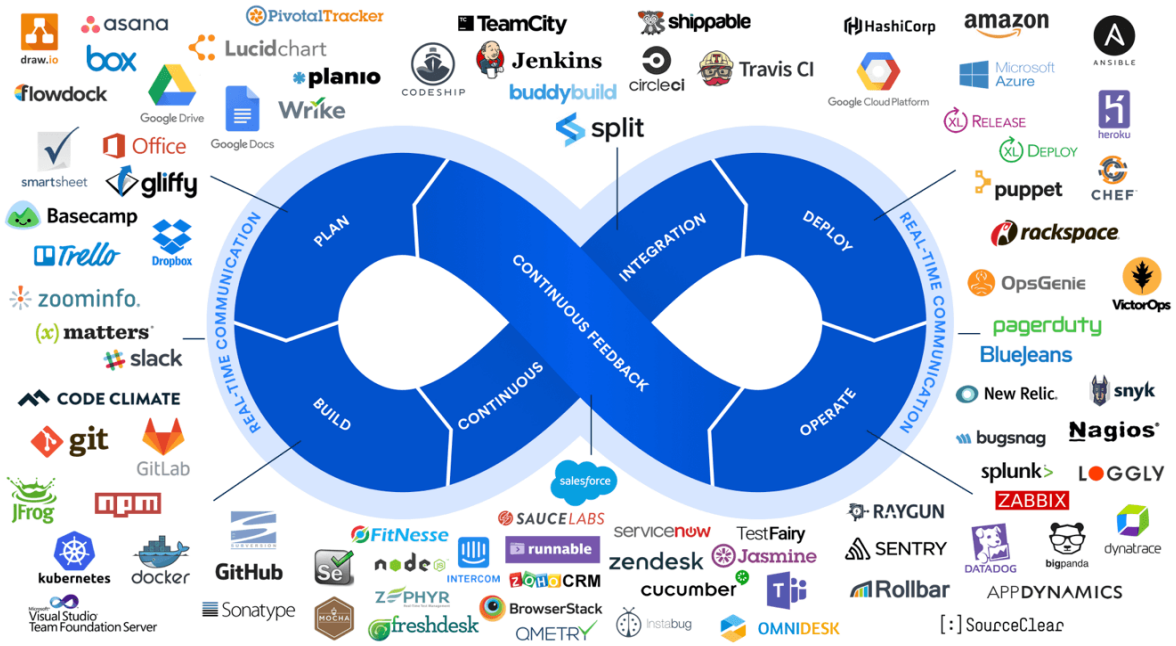
JFrog Artifactory is a tool designed to store the binary output of the build process for use in distribution and deployment. Artifactory provides support for several of package formats such as Maven, Debian, NPM, Helm, Ruby, Python, and Docker. JFrog offers high availability, replication, disaster recovery, scalability, and works with many on-premise and cloud storage offerings.

This for sure is not a comprehensive list. There are constantly new tools being developed in both the configuration orchestration and management landscape.

6.1.4 Other Tools

HO-6.1.4	Perform a guided exercise to search for most updated existing tools on the market for DevOps, including those which support specific stages, as well as tools that are suites and providing holistic solutions (HO-1)

DevOps brings an overall vision of / perspective on the variety of tools that exist on the market, and which are supporting its holistic approach.
The following image may describe this variety of tools, and its continuous update.



References

This document has been designed and created utilizing first-hand experiences gathered from the industry by the SMEs involved in creating DevOps United.

- A Taxonomy for Learning, Teaching, and Assessing: A Revision of Bloom's Taxonomy of Educational Objectives — L. Anderson, P. W. Airasian, and D. R. Krathwohl (Allyn & Bacon 2001)
- Revised Bloom's Taxonomy Action Verbs. Available at https://www.apu.edu/live_data/files/333/blooms_taxonomy_action_verbs.pdf
- **[DevOps for the Modern Enterprise]** Winning Practices to Transform Legacy IT Organizations — Mirco Hering (April 2018)
- **[The DevOps Handbook]** How to Create World-Class Agility, Reliability, & Security in Technology Organizations — Gene Kim, Jez Humble, John Willis & Patrick Debois (October 2016 edition)
- **[Embedding DevOps in the Enterprise]** Cutter IT Journal (November 2011 edition)
- **[DevOps Guide]** The IT Revolution (2015 edition)
- **[DevOps for Dummies]** IBM — Sanjeev Sharma & Bernie Coyne (John Wiley & Sons, 2nd edition 2015)
- **[ISTQB-FL 2018]** ISTQB Foundation Level Syllabus version 2018. Available at https://www.istqb.org/downloads/category/51-ctfl2018.html
- **[Agile Alliance organization]** https://www.agilealliance.org