Mu.SA
museum sector alliance

Solution Deployment:
Deploying Software

Filipe Correia, UPorto

Co-funded by the Erasmus+ Programme of the European Union

Aim and objectives

The aim of this presentation is to present the main concepts about *software deployment*, including its connection with DevOps principles and practices, the associated packaging and distribution methods, the relation between deployment and software architecture, and the activities and technologies used for deploying.

Co-funded by the Erasmus+ Programme of the European Union

Learning outcomes

At the end of this presentation, you will be able to:

- Identify software packaging and distribution methods;
- Explain the relationship between deployment and software architecture;
- Identify technologies and standards used for deploying software;
- Design the deployment environment of a given system.

Co-funded by the Erasmus+ Programme of the European Union

Table of contents

Co-funded by the
Erasmus+ Programme
of the European Union

Context

- Historically, new software releases would be **sparse**, include a **substantial number of features**, and require **manual installation** on each device where it was intended to run;
- **Deploying** software has changed substantially with the Internet and the increased use of distributed systems. Software is now often deployed to **many customers** in minutes on a schedule that is controlled mostly by its producer;
- **Cloud computing** democratized the access by software developers to computing **infrastructures** that can **scale** according to the needs of any organization, and avoid the **risks** and **cost** of large investments in infrastructure.

Co-funded by the Erasmus+ Programme of the European Union

Mu.SA museum sector alliance

Packaging and Distribution

In order to distribute a software system it is often beneficial to *package* it in some way. Popular approaches to packaging software include *installers* and *package managers*.

- **Installers:** Software that installs another software system onto a computer. Some installers will copy to the device files that they already contain, while others will download the required files from a remote location.

  Examples: Windows Installer – MSI , Android Application Package – APK.

(continues)

Co-funded by the Erasmus+ Programme of the European Union

Packaging and Distribution

- **Package managers:** Package managers are software tools used to manage the installation of other software systems. They automate the processes of installing, upgrading, configuring and removing software for a device in a consistent and convenient way, from a repository of available software *packages*.

  Examples: Microsoft Store (Windows), App Store (Mac), Homebrew (Mac) Advanced Package Tool – APT (Linux/Debian).

Co-funded by the Erasmus+ Programme of the European Union

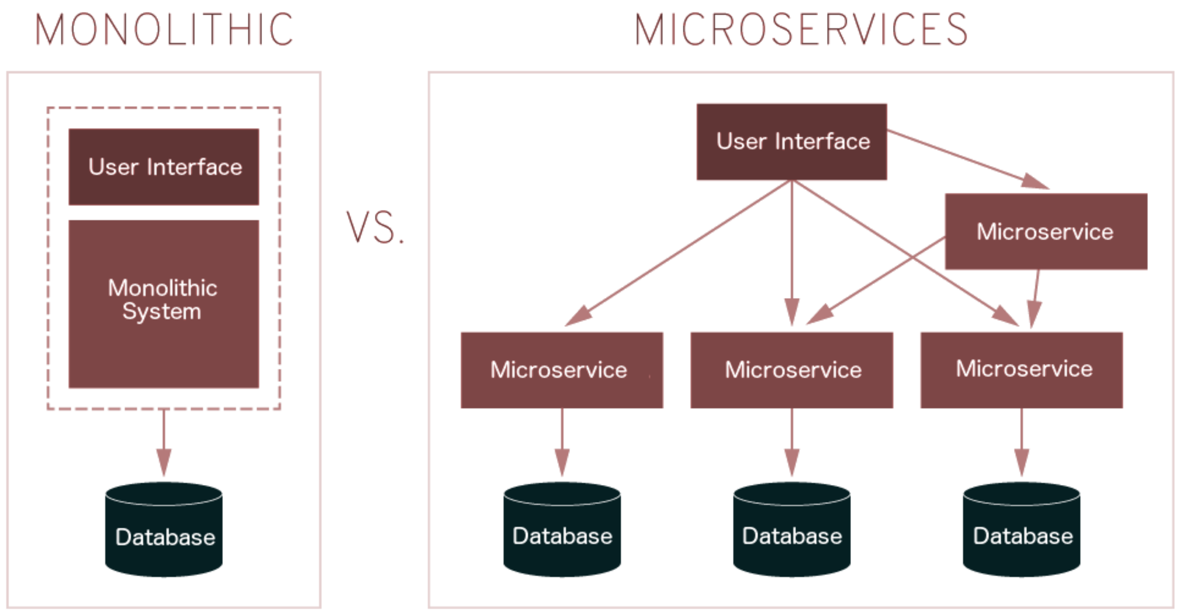
Software Architecture

- Cloud computing allows the flexibility to *scale* a system to follow variation in its usage;
- However, the architecture of software systems needs to be designed specifically to take full benefit of the scalability supported by cloud services:
  - **Monolithic architectures** are being used when the system consists of a single software service and is deployed and run as a single unit.
  - **Microservices architectures** are used when a system is designed to have many different services that can be deployed and run independently, and that are resilient to failures from other services.

Co-funded by the Erasmus+ Programme of the European Union

Software Architecture

- The figure below illustrates the differences between the two kinds of architecture: monolithic and microservices;



Co-funded by the Erasmus+ Programme of the European Union

Software Architecture

- Both architectures (monolithic and microservices) can organize the system according to different components (e.g. the monolithic system in the previous slide features two components) but the monolithic system will run these as a **single service**, while microservices will favor having **multiple services**, each with its own database, and they should be able to evolve almost independently;

Co-funded by the Erasmus+ Programme of the European Union

Activities

- Software deployment comprises a set of activities required to make a software system available to its intended end-users;

- These activities include:
  - **Build and Release:** *Building* software involves the several activities required to generate a runnable software system from source code. These so called *builds* are designated as *releases* when they are made available to the general public;
  - **Install:** Making the software system available in the device where it will be run. This often goes beyond copying files to a device, and implies also adding configuration parameters to the software system or to the underlying operating system;

Co-funded by the Erasmus+ Programme of the European Union

Activities

- **Run:** To *run* a software system is to *execute* it in a chosen hardware device, thus making use of its processor, but possibly also memory and storage resources;
- **Update:** To update a software system is to replace it by a newer release. On some cases this can be done manually by the end-user; in other cases it can be done automatically or semi-automatically;
- **Auto-update:** A software system with auto-update capability is one that is able to detect that a new release exists, and replace itself with that new release, with no intervention by the end-user;

Co-funded by the Erasmus+ Programme of the European Union

Activities

- **Schema / data migration:** Often the structure of data from the previous version of a software system needs to be transformed so that it can be used used by the new version. Schema/data migration is the process of doing such data transformations, and is performed on a database whenever it is necessary to update or revert it to some newer or older version;
- **Decommission:** Decommissioning a software system is to stop its execution and remove it from service.

Co-funded by the Erasmus+ Programme of the European Union

Automation

Software deployment has changed through the years to respond to the ever-more demanding needs of organizations. Although it is done **manually** in a lot of contexts, there are now many others in which **continuous deployment** is the norm. These two different approaches can be defined in the following way:

- **Manual Deployment**
  The process of putting new versions of a software system into operation for end-users by manually copying the software to its intended infrastructure and running it;

- **Continuous Deployment**
  Teams produce software in small batches, that are delivered to the customer or end-users as as soon as they are ready. This approach fosters an incremental approach to updating applications. It requires that the deployment process is automated, but helps to reduce the cost and risk of delivering a new version.

Co-funded by the Erasmus+ Programme of the European Union

Environments

- A deployment environment is composed by the **infrastructure** and the **software** in which a software system is deployed and executed;
- Different kinds of deployment environments serve different purposes, from *testing* to *staging*, to *production*:
  - A **testing** environment has the goal of allowing human testers to exercise new or updated features before they are incorporated into other environments;

(continues)

Co-funded by the Erasmus+ Programme of the European Union

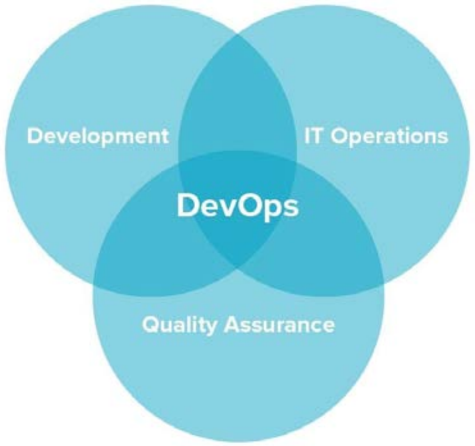
Environments

- A **staging** environment seeks to provide access to the next immediate version of a system, in an environment that is as similar as possible to the one in production, in order to allow trying and experimenting with that version before it goes into production;
- A **production** environment is the one that users directly interact with. Deploying to production is in many contexts a sensitive step, especially when the system can’t appear to stop from the end-user perspective.

Co-funded by the Erasmus+ Programme of the European Union

DevOps

- The word *DevOps* is the agglutination of the words *Development* and *Operations*. DevOps is a movement based in the idea of joining professional roles that traditionally would have little interaction with each other and that span *software development*, *quality assurance* and *operations*.
- **Software Development** – the activities related with the actual creation of the software system;
- **Quality Assurance** – testing and other activities that prevent software defects or other problems that may directly affect customers;
- **Operations** – the activities related with putting and keeping software in operation, from infrastructure, to deployment, to monitoring;



Co-funded by the Erasmus+ Programme of the European Union

DevOps

- The core values of DevOps can be enounced as the CAMS acronym, which stands for:
    - ***C**ulture* – At the core of DevOps are motivations related to people and business. Fostering a culture of sharing responsibility and multidisciplinary teams is one of the main goals of DevOps;
    - ***A**utomation* – The automation of processes allows to make them repeatable, which eases debugging and reduces the possibility of human errors;
    - ***M**easurement* – Monitor running systems to obtain feedback that can inform improvements done by software development teams;
    - ***S**haring* – DevOps values transparency and openness. The collective intelligence of a team makes it more efficient and greater than just the sum of its parts.

Co-funded by the Erasmus+ Programme of the European Union

DevOps & Deployment

- DevOps has several implications in what concerns the deployment of software;
- The **automation** of everything related to deployment is one of the most important implications, and can span different concerns:
    - **Infrastructure as code** – defining the infrastructure needed by a system using code allows to recreate at any time using a cloud service.

      Tool examples: Chef, Puppet, Terraform

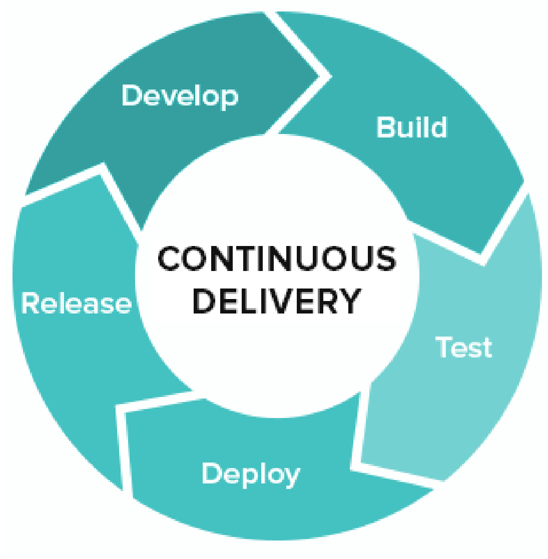
(continues)

Co-funded by the Erasmus+ Programme of the European Union

DevOps & Deployment

- **Continuous delivery** – streamlining the process of incorporating new development in a production system allows software to be released at any time and as often as needed.

  Tool examples: Jenkins, Travis, Gitlab pipelines



(continues)

Co-funded by the Erasmus+ Programme of the European Union

Mu.SA museum sector alliance

DevOps & Deployment

- **Test automation** – automating the execution of tests provides a safety net that minimizes concerns when making changes to the system.
  Tool examples: xUnit, Selenium, Cucumber
- **Containerization** – Software containers allow to isolate the system from the environment in that it is deployed, so that it can be easily deployed to any environment as needed.
  Tool examples: Docker

Co-funded by the Erasmus+ Programme of the European Union

DevOps & Deployment

- **Orchestration** – Automated configuration and coordination of computer systems and software, often with the use of containerization.
  Tool examples: Kubernetes, Docker swarm
- **Telemetry** – Instrumentation of software systems to collect data on their use and performance (e.g., how are feature used, crashes and start-up and processing times).
  Tool examples: Datadog, Kibana

Co-funded by the Erasmus+ Programme of the European Union