1. **De ce avem nevoie de Jenkins ?**

Jenkins este un server de automatizare autonom, open-source, care poate fi folosit pentru a automatiza tot felul de sarcini legate de construirea, testarea și livrarea sau implementarea software-ului.

1. **Ce este un Jenkins pipeline ?**

Pipeline este o secvență de pași controlați de o logică definită - care coordoneaza activități de lungă durată care se pot întinde pe mai mulți agenți de build.

1. **Care sunt tipurile de pipeline Jenkins ?**

Tipuirile de pipeline in jenkins sunt: declarative sau scriptate.

Cele declarative folosesc numeroase etape de construire, generice, predefinite (adică fragmente de cod) pentru a face build în funcție de nevoile noastre

La cele scriptate, etapele pot fi definite la comandă și utilizate folosind o aplicație groovy. sintaxă care oferă un control mai bun și niveluri de execuție

1. **Care sunt modalitățile de a declanșa un pipeline Jenkins ?**

Un Pipeline poate fi creat prin una din metodele acestea:

* [Through Blue Ocean](https://www.jenkins.io/doc/book/pipeline/getting-started/#through-blue-ocean) - după configurarea unui proiect Pipeline în Blue Ocean, interfața de utilizare Blue Ocean vă ajută să scrieți fișierul Jenkins al Pipeline-ului și să îl trimiteți controlului sursei.
* [Through the classic UI](https://www.jenkins.io/doc/book/pipeline/getting-started/#through-the-classic-ui) - puteți introduce un pipeline de baza direct în Jenkins prin interfața de utilizare clasică.
* [In SCM](https://www.jenkins.io/doc/book/pipeline/getting-started/#defining-a-pipeline-in-scm) - puteți scrie manual un fișier Jenkins, pe care îl puteți commit’e în repozitoriul sursă al proiectului.

1. **Cum se realizează integrarea continuă cu Jenkins?**

Integrarea continuă este un proces în care modificările codului unui dezvoltator sunt integrate constant în codul principal și acestea vor fi testate automat, iar rezultatele testelor vor decide dacă modificarea este gata pentru implementare. In acest proces -

-Dezvoltatorul face o modificare - commit/pull\_request - în ramura caracteristică

- Sistemul de management al controlului sursei generează evenimente adecvate

-Pluginurile Jenkins specifice SCM, cum ar fi Git/SVN, vor detecta acele evenimente din depozitele configurate și aceste evenimente vor fi folosite pentru a declanșa - construi/dependent/test - joburi pe Jenkins

-După finalizarea lucrărilor de testare/dependente, modificarea/patch-ul va fi etichetat în funcție de starea jobului de testare

-Pe baza stării (adică pregătirea unei schimbări pentru a fi fuzionată cu filiala principală), strategia/instrumentul de livrare continuă sau implementare continuă o va duce mai departe.

1. **Pentru ce este nevoie de variabile de mediu în Jenkins ?**

Variabila de mediu Jenkins este o variabilă globală expusă prin variabila env și utilizată oriunde în fișierul Jenkins. Orice valoare stocată în variabila env este stocată ca tip String. Variabilele de mediu pot fi setate fie la nivelul superior al conductei, la nivelul etapei specifice, fie în interiorul blocului de script

1. **Numiți 5 variabile de mediu implicit disponibile în Jenkins**

BRANCH\_NAME,

BUILD\_NUMBER,

BUILD\_TAG, BUILD\_URL,

WORKSPACE

1. **Ce se înțelege prin integrare continuă (Continuous Integration) ?**

Integrarea continuă este procesul de verificare continuă a codului dezvoltatorului într-un sistem de control al versiunilor și de declanșare a construcției pentru verificarea și identificarea erorilor din codul scris.

Acesta este un proces foarte rapid și, de asemenea, le oferă șansa de a remedia erorile. Jenkins este un astfel de instrument de integrare continuă.

1. **Ce se înțelege prin livrare continuă (Continuous Delivery) ?**

**???**

1. **Ce se înțelege prin plasare continuă (Continuous Deployment) ?**

**???**

1. **Care sunt diferențele dintre integrarea continuă, livrarea continuă și plasare continuă ?**

• Dezvoltatorii care practică integrarea continuă fac “merge” la modificările în ramura principală cât mai des posibil. Procedând astfel, spam-ul de merge’uri care se întâmplă de obicei atunci când oamenii așteaptă ziua lansării pentru a face merge la modificările lor în ramura de principala.

• Livrarea continuă este o extensie a integrării continue pentru a vă asigura că puteți introduce rapid noi modificări clienților dumneavoastră într-un mod durabil. Aceasta înseamnă că, pe lângă faptul că v-ați automatizat testarea, ați automatizat și procesul de lansare și vă puteți implementa aplicația în orice moment făcând clic pe un buton.

• Implementarea continuă merge cu un pas mai departe decât livrarea continuă. Cu această practică, fiecare modificare care trece prin toate etapele pipeline-ului de producție este transmisă clienților tăi. Nu există nicio intervenție umană și doar un test eșuat va împiedica implementarea unei noi modificări în producție.

1. **Care sunt factorii de succes pentru integrarea continuă ?**

* Mentineti un repozitoriul cu cod
* Automatizati build’ul
* Faceti ca build’ul sa faca teste automat
* Toate commit’urile sa respecte linia de baza
* Toate commit’urile trebuie construite (build)
* Build’ul sa fie rapid
* Testati intr-o clona a repozitoriului mediu de productie
* Faceti sa fie usor preluarea ultimilor modificari
* Toti trebuie sa vada rezultatele ultimului build
* Automatizati implementarea

1. **Care este rolul serverului de integrare continuă ?**

Funcția serverului CI este de a integra în mod continuu toate modificările făcute și trimise în repozitoriu de către diferiți dezvoltatori și de a verifica erorile de compilare. Trebuie să construiască codul de mai multe ori pe zi, de preferință după fiecare comitere, astfel încât să poată detecta ce comitere a provocat ruperea dacă ruperea are loc.

1. **Cum DevOps influențează dezvoltarea de software ?**

DevOps aduce beneficii în gestionarea lansării software-ului unei organizații prin standardizarea mediului de dezvoltare. Evenimentele pot fi urmărite mai ușor și pot fi activate procese de management documentate și rapoarte detaliate. Abordarea DevOps oferă dezvoltatorilor mai mult control asupra mediului, oferind infrastructurii o înțelegere mai centrată a aplicație.

1. **Numiți 5 instrumente CI/CD**

* Bamboo
* Jenkins
* Circle**ci**
* TeamCity
* Codeship

1. **Care este ciclul de viață al unui produs program ?**
2. **Edit time** is when the [source code](https://en.wikipedia.org/wiki/Source_code) of the program is being edited. This spans initial creation to any [bug](https://en.wikipedia.org/wiki/Software_bug) fix, [refactoring](https://en.wikipedia.org/wiki/Code_refactoring), or addition of new [features](https://en.wikipedia.org/wiki/Software_feature). Editing is typically performed by a person, but automated [design tools](https://en.wikipedia.org/wiki/Programming_tool) and [metaprogramming](https://en.wikipedia.org/wiki/Metaprogramming) systems may also be used.
3. [**Compile time**](https://en.wikipedia.org/wiki/Compile_time) is when source code is translated into [machine code](https://en.wikipedia.org/wiki/Machine_code) by a [compiler](https://en.wikipedia.org/wiki/Compiler). Part of this involves language checking, such as ensuring proper use of the [type system](https://en.wikipedia.org/wiki/Type_system). The result of a successful compilation is an [executable](https://en.wikipedia.org/wiki/Executable).
4. [**Link time**](https://en.wikipedia.org/wiki/Link_time) connects all of the necessary machine code components of a program, including externals. It is very common for programs to use functions implemented by external [libraries](https://en.wikipedia.org/wiki/Library_(computing)), all of which must be properly linked together. There are two types of linking. [Static linking](https://en.wikipedia.org/wiki/Static_linking) is when the connection is made by the compiler, which is always prior to execution. [Dynamic linking](https://en.wikipedia.org/wiki/Dynamic_linking), however, is performed by the [operating system](https://en.wikipedia.org/wiki/Operating_system) (OS) just before, or even during, execution.
5. **Distribution time** is the process of transferring a copy of a program to a user. The distribution format is typically an executable, but may also be source code, especially for a program written in an [interpreted language](https://en.wikipedia.org/wiki/Interpreted_language). The means of distribution can be physical media such as a [USB flash drive](https://en.wikipedia.org/wiki/USB_flash_drive) or a remote download via the [Internet](https://en.wikipedia.org/wiki/Internet).
6. [**Installation time**](https://en.wikipedia.org/wiki/Installation_(computer_programs)) gets the distributed program ready for execution on the user's computer, which often includes storing the executable for future loading by the OS.
7. [**Load time**](https://en.wikipedia.org/wiki/Load_time) is when the OS takes the program's executable from storage, such as a [hard drive](https://en.wikipedia.org/wiki/Hard_disk_drive), and places it into active [memory](https://en.wikipedia.org/wiki/Computer_memory), in order to begin execution.
8. [**Run time**](https://en.wikipedia.org/wiki/Run_time_(program_lifecycle_phase)) is the execution phase, when the [central processing unit](https://en.wikipedia.org/wiki/Central_processing_unit) executes the program's machine code instructions. Programs may run indefinitely. If execution terminates it will either be normal, expected behavior or an abnormality such as a [crash](https://en.wikipedia.org/wiki/Crash_(computing)).

**AICI NE UVEREN CA ESTE CORECT!!!!! + Trebuie traduse in romana**

1. **Ce reprezintă și care este rolul testării unitare ?**

Testarea unitară implică testarea componentelor individuale ale programului sau aplicației software. Scopul principal din spatele acestui lucru este de a verifica dacă toate piesele individuale funcționează așa cum este prevăzut. O unitate este cunoscută drept cea mai mică componentă posibilă a software-ului care poate fi testată.

1. **Numiți 5 cadre de testare bazate pe xUnit**

**JAVA:** Artos, Agitar etc.

**Python:** unittest, pytest etc.

**C++:** Cput, API Sanity Checker etc.

**Bash:** shUnit2

**PHP:** PHPUnit, Lens, Tester etc.

**MATLAB:** mlUnit, matlab.unitest, MOxUnit etc.

1. **Care este rolul aserțiunilor în testarea automată ?**

Assertiunea este o colecție de metode utilitare care sprijină afirmarea condițiilor în teste.

1. **Cum Jenkins identifică schimbarea versiunii în repozitoriu ?**

**Sunt 2 metode, eu licino nu le stiu.**