

Jenkins

Ce este devops?

DevOps este un set de practici care combină dezvoltarea de software și partea de operation. Acesta își propune să scurteze ciclul de viață al dezvoltării sistemelor și să ofere livrare continuă cu o calitate înaltă a software-ului .

Înainte de devops procesul de livrare a produsul era mult mai lent și se folosea conceptul de monolit în multe foarte proiecte. De asemenea, toate cerințele produsului trebuiau realizate la un moment fix, nu se mai faceau de obicei modificări ulterioare și nu se lucra în paralel. După faza de planificare se trecea la dezvoltare, apoi la testare și apoi la construirea produsului final. Acest proces de construire era astfel mult mai lent deoarece nu exista flexibilitate pentru modificarea cerințelor după încheierea fazei de planificare.

De asemenea nu există un mecanism integrat, echipa de development putea lucra în alt mediu, pe când echipa de operations în alt mediu. Ce probleme pot reieși de aici ?



Ce este CI ?

CI (**continuous integration**) este o practică modernă de dezvoltare de software în care modificările incrementale ale codului sunt făcute frecvent, în mod fiabil și la final scopul este să obținem un artefact.

Tool-uri de CI/CD:

Jenkins

Azure devops

Code pipeline

[GitLab](#) CI/CD -: de citit



Diferența continuous deployment și delivery

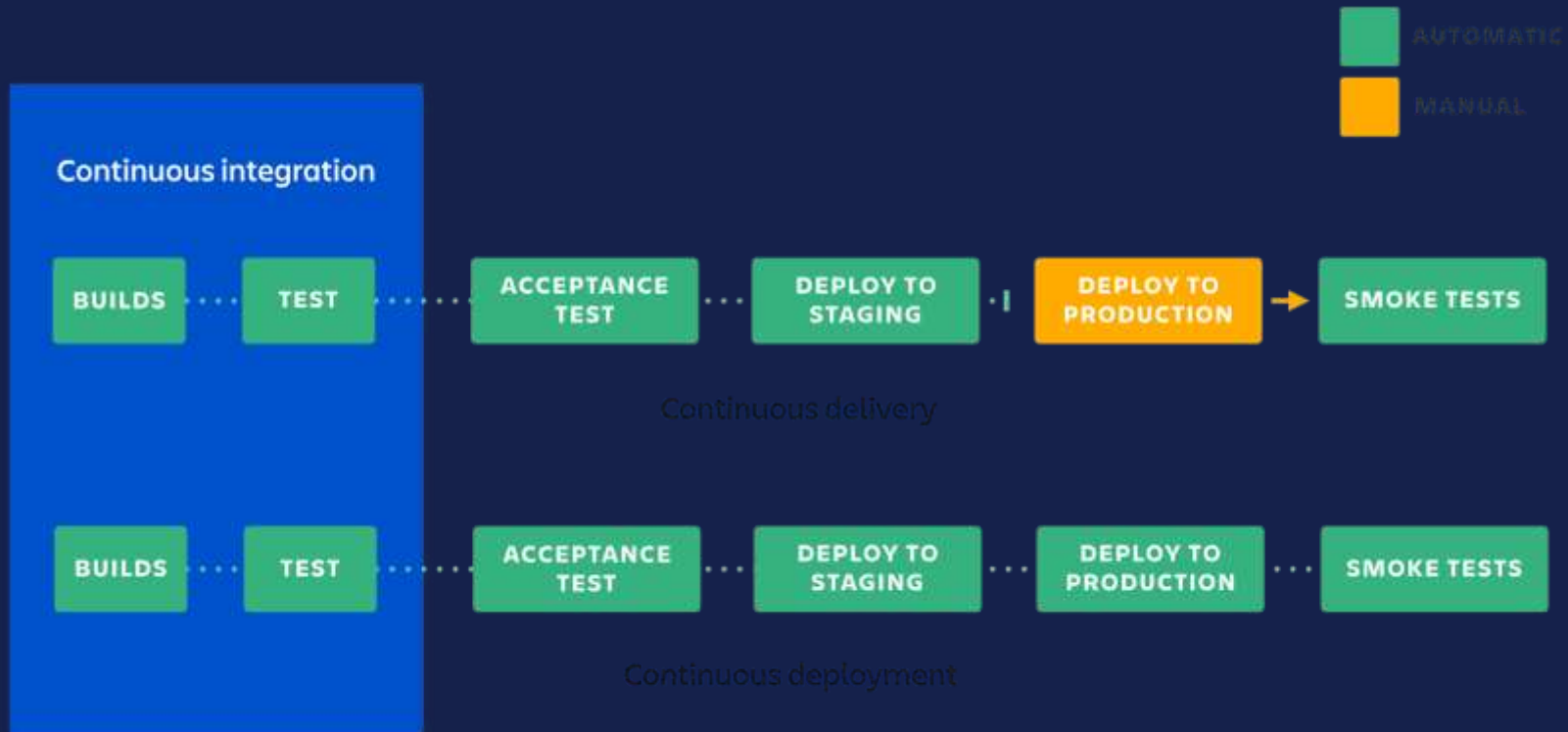
Scopul principal al livrării de software este să reușim să punem acele idei și pe producție.

Continous deployment - Acest model se foloseste de automatizare pentru a face un deploy pe diferite environment-uri (de la uat, non-prod, beta și până la producție). Practic se asigura, că într-un mod automatizat, aplicația noastră ajunge pe acele environment-uri

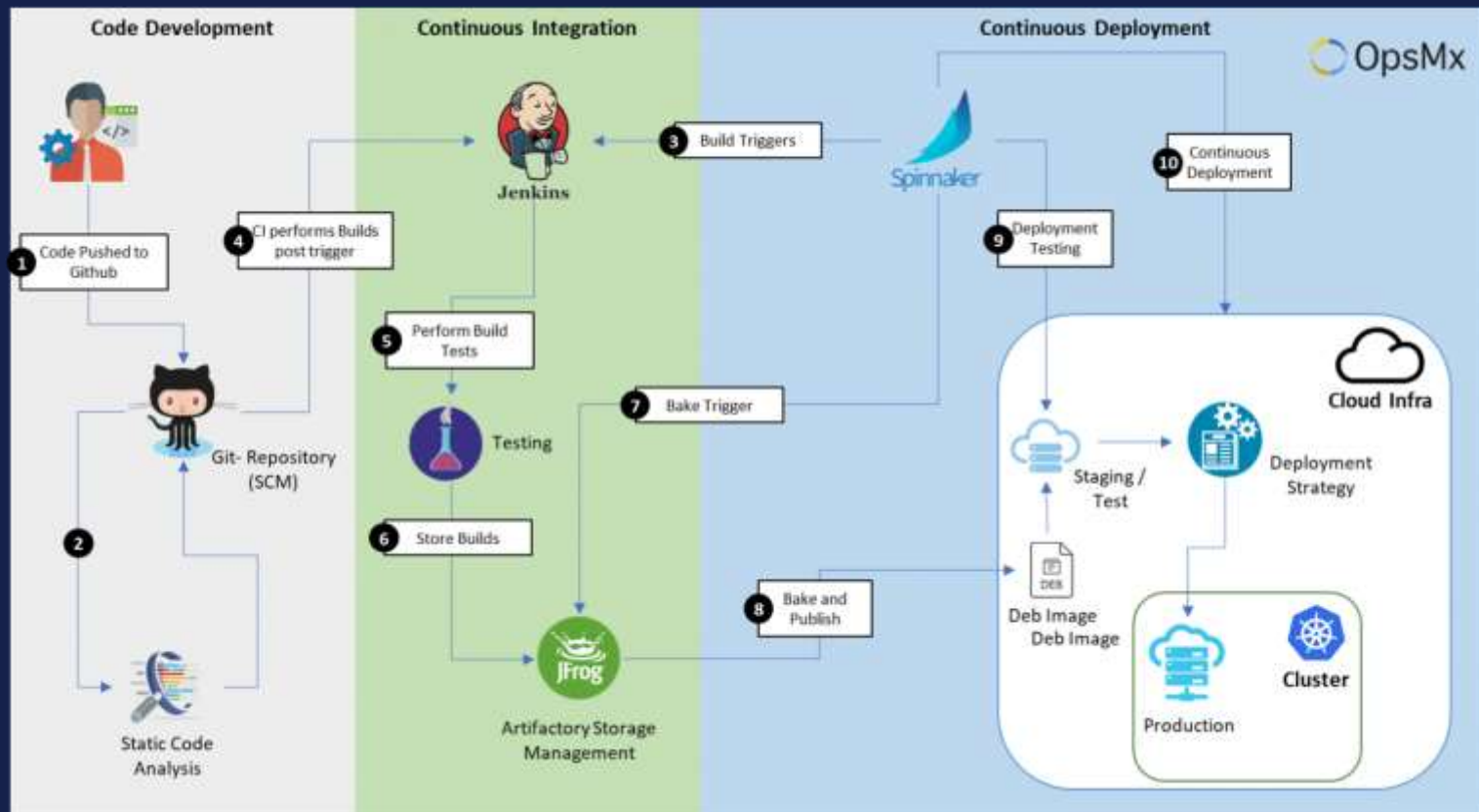
Continuous Delivery – Reprezintă pașii automatizați pentru a duce schimbările pe producție. Pe când **Continous deployment** se axează pe deploy în sine, continous delivery se axează pe strategiile de release. Dupa ce se fac verificările de cod, se confirmă manual de către cel responsabil și apoi noua modificare se duce direct pe producție.



Diferența continuous deployment și delivery

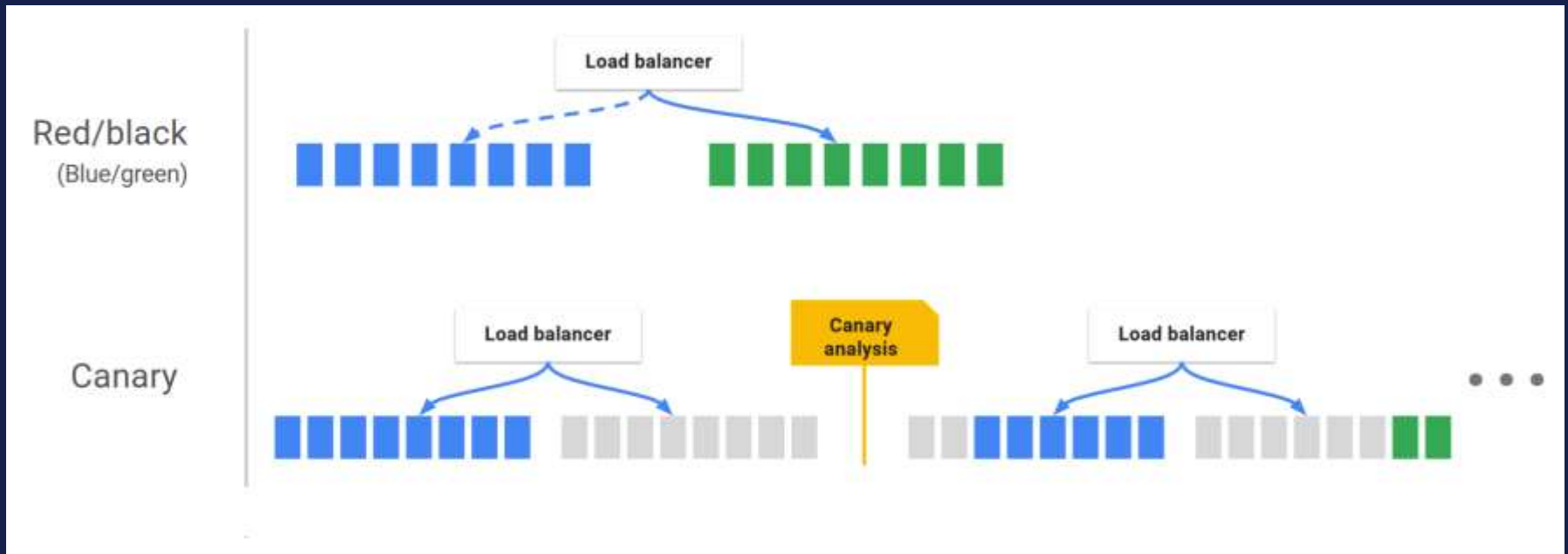


Ce este CI/CD ?



Sursă [image](#)

Canary vs blue-green deployment



Ce este jenkins ?

Jenkins este un tool de automatizare open-source scris în ce este folosit în devops pentru CI/CD (**integrare continuă / livrare continuă**). Astfel, le permite dezvoltatorilor să automatizeze procesul de construire, testare și implementare a codului lor.

Este foarte util deoarece are numeroase plugin-uri pentru majoritatea tool-urilor create de comunitate .



Instalare jenkins pe Ubuntu

1. Prima dată este nevoie să instalăm java:

```
sudo apt update  
sudo apt install openjdk-8-jdk
```

2. Add the Jenkins Repository

```
curl -fsSL https://pkg.jenkins.io/debian-stable/jenkins.io.key | sudo tee /usr/share/keyrings/jenkins-keyring.asc > /dev/null
```

```
echo deb [signed-by=/usr/share/keyrings/jenkins-keyring.asc] https://pkg.jenkins.io/debian-stable binary/ |  
sudo tee /etc/apt/sources.list.d/jenkins.list > /dev/null
```

3. Install Jenkins

```
sudo apt update  
sudo apt install jenkins
```



Instalare jenkins pe Ubuntu

Verificarea statusului: **sudo systemctl status jenkins**

În mod normal ar trebui să fie activ. Jenkins rulează pe portul 8080, în caz de firewall-ul este activat, este recomandat să se ruleze comanda: **sudo ufw allow 8080**

Pentru verificarea statusului firewall-ului: **sudo ufw status**

Cum se verifică dacă funcționează?

Se deschide un browser în serverul virtual ubuntu unde am instalat și accesăm:

<http://localhost:8080>

Pașii de instalare se pot vedea aici: [Install steps](#)



Instalare jenkins pe Docker

Pentru a crea un volum:

```
docker volume create jenkins
```

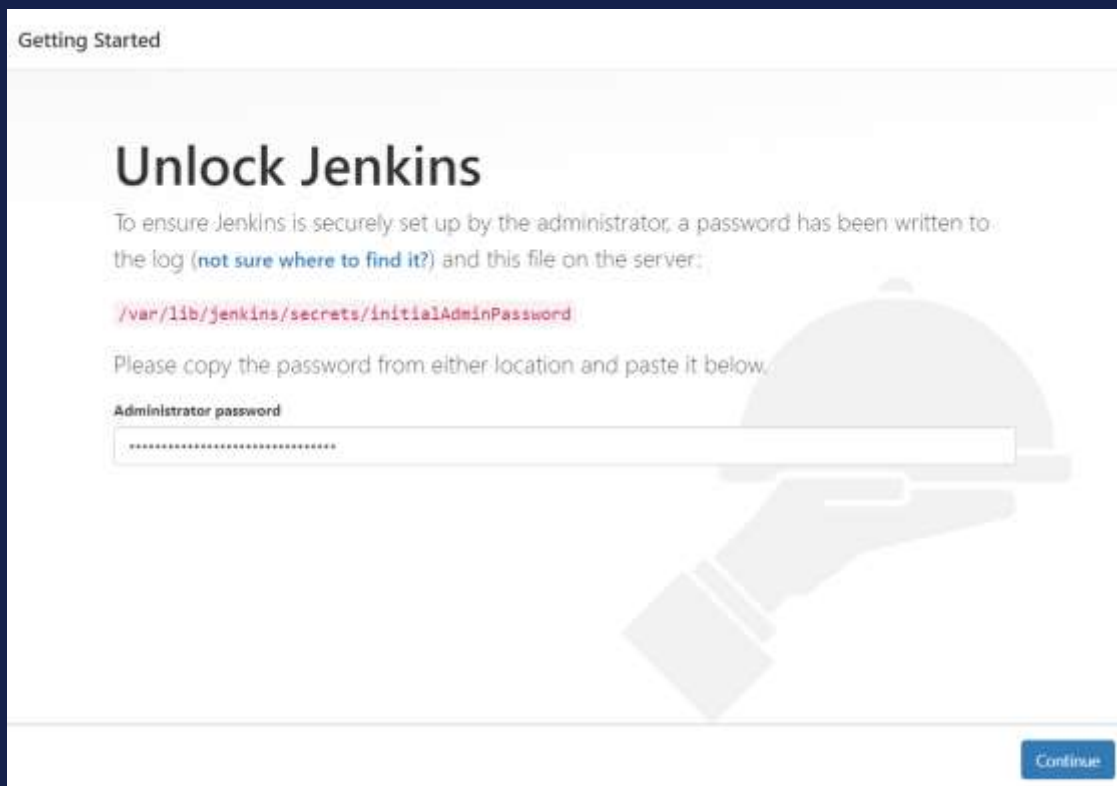
Pentru a porni containerul:

```
docker run -d --name myjenkins -p 8083:8080 -p 50000:50000 -v  
jenkins:/var/jenkins_home jenkinsci/blueocean:1.25.6
```



Configurarea jenkins

După instalare accesați **http://localhost:8080/** și va cere un token care este salvat în configurația de jenkins. Copiați tokenul ce a fost generat în **/var/lib/jenkins/secrets/initialAdminPassword** și apăsați continue .



Getting Started

Unlock Jenkins

To ensure Jenkins is securely set up by the administrator, a password has been written to the log (not sure where to find it?) and this file on the server:

```
/var/lib/jenkins/secrets/initialAdminPassword
```

Please copy the password from either location and paste it below.

Administrator password

Continue

Configurarea jenkins

În jenkins putem adauga mai multe plugin-uri. Acestea ajută la integrarea prin jenkins a diferite alte programe sau tool-uri. Este recomandat ca o parte dintre aceste plugin-uri să se instaleze la prima pornire a jenkins prin **“install suggested plugins”** .



Configurarea jenkins

Getting Started

Create First Admin User

Username:

Password:

Confirm password:

Full name:

E-mail address:

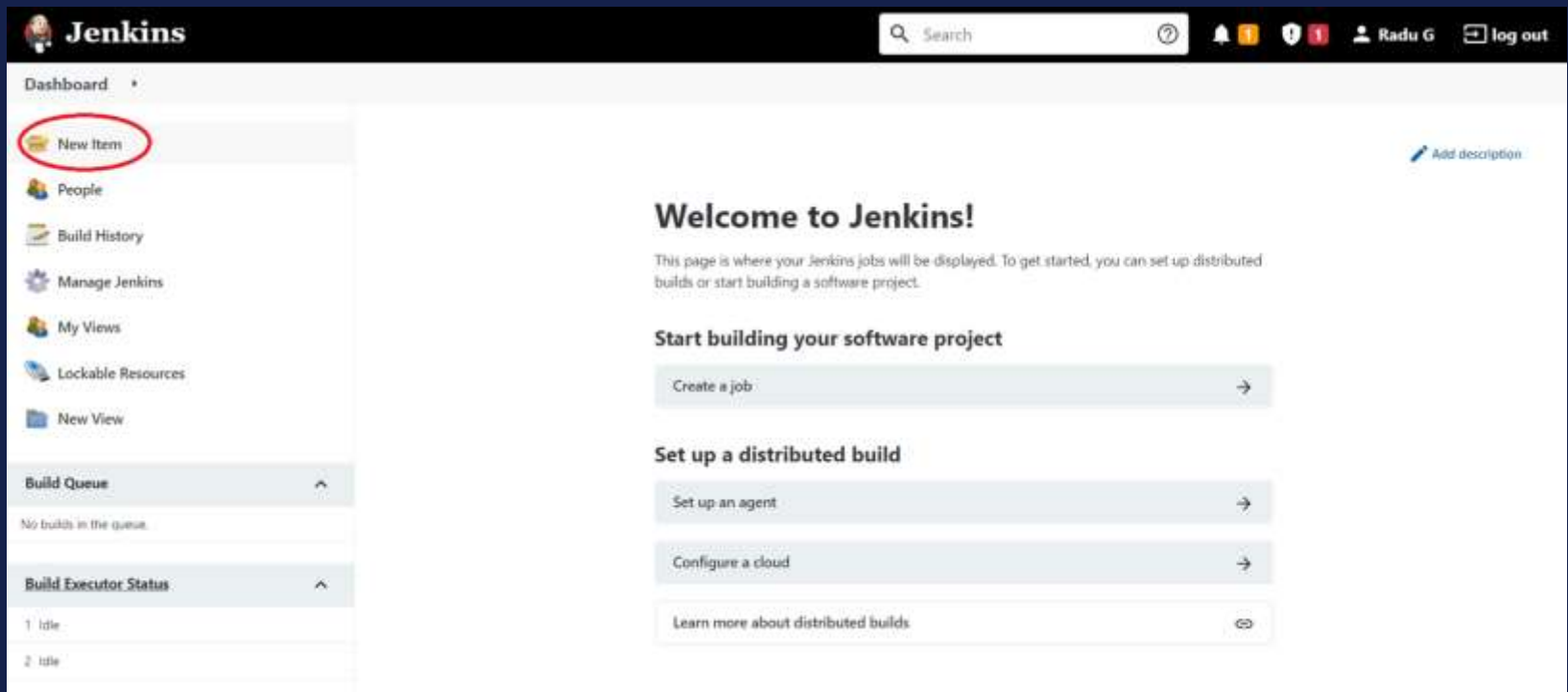
Jenkins 2.319.3

[Skip and continue as admin](#) [Save and Continue](#)

Completați câmpurile lipsă. Userul și parola sunt importante deoarece vi se vor cere la logare. După introducerea datelor apăsați pe **“Save and continue”** și **“Start using jenkins”**.

Primul job în jenkins

Un job în jenkins este un task care se poate rula pentru a atinge un obiectiv cerut .




Primul job în jenkins

Enter an item name


FIRST_JOB

* Required field




Freestyle project

This is the central feature of Jenkins. Jenkins will build your project, combining any SCM with any build system, and this can be even used for something other than software build.




Pipeline

Orchestrates long-running activities that can span multiple build agents. Suitable for building pipelines (formerly known as workflows) and/or organizing complex activities that do not easily fit in free-style job type.




Multi-configuration project

Suitable for projects that need a large number of different configurations, such as testing on multiple environments, platform-specific builds, etc.




Folder

Creates a container that stores nested items in it. Useful for grouping things together. Unlike view, which is just a filter, a folder creates a separate namespace, so you can have multiple things of the same name as long as they are in different folders.



Multibranch Pipeline

Creates a set of Pipeline projects according to detected branches in one SCM repository.



Organization Folder

Creates a set of multibranch project subfolders by scanning for repositories.

OK

Primul job în jenkins

În acest prim job vom executa un mesaj simplu de tip 'hello world' pe care îl vom salva într-un fișier text .

```
pwd
```

```
echo "hello world from jenkins" > jenkins_file.txt
```

```
ls
```

Putem executa cu job-ul apăsând **build now** .

Primul job în jenkins

De asemenea, putem salva valori în variabile și a le folosi printr-un job:

```
Age=21  
echo "I have $Age years old"
```

Dacă putem executa orice comenzi de shell, înseamnă că putem scrie script-uri, dar și că le putem executa , ceea ce ne conferă foarte multă flexibilitate .

Parametrii

Un parametru în Jenkins ne permite să transmitem date în joburile de Jenkins. Acest lucru se poate realiza din setările 'generale' ale unui job selectând '**this project is parameterized**' .

Ca și în cazul de mai devreme putem selecta o variabilă cu numele "Age" și să punem o valoare default.

Diferența este faptul că atunci când vom rula job-ul vom putea introduce valoarea dorită și aceasta se va transmite mai departe ca parametru pentru job-ul nostru .

De asemenea, putem avea mai multe tipuri de variabile de exemplu: **boolean, choice sau password**.



Variabile build in

Jenkins oferă un set de variabile “build in”, adică ele nu trebuie definite înainte și care ne pot ajuta atunci când rulăm un job. Lista acestor variabile poate fi găsită [aici](#).

Exemple:

BUILD_NUMBER - 6

BUILD_ID - 7

JOB_NAME - mytestjob

http://Your_jenkins_IP:8080/env-vars.html/

Variabile build in

Este util dacă vrem de exemplu să facem un backup în fiecare zi la o bază de date în mod automat prin jenkins. Fie că salvăm acel job într-o locație externă **AWS S3** sau pe serverul local, este important să le diferențiem cumva.

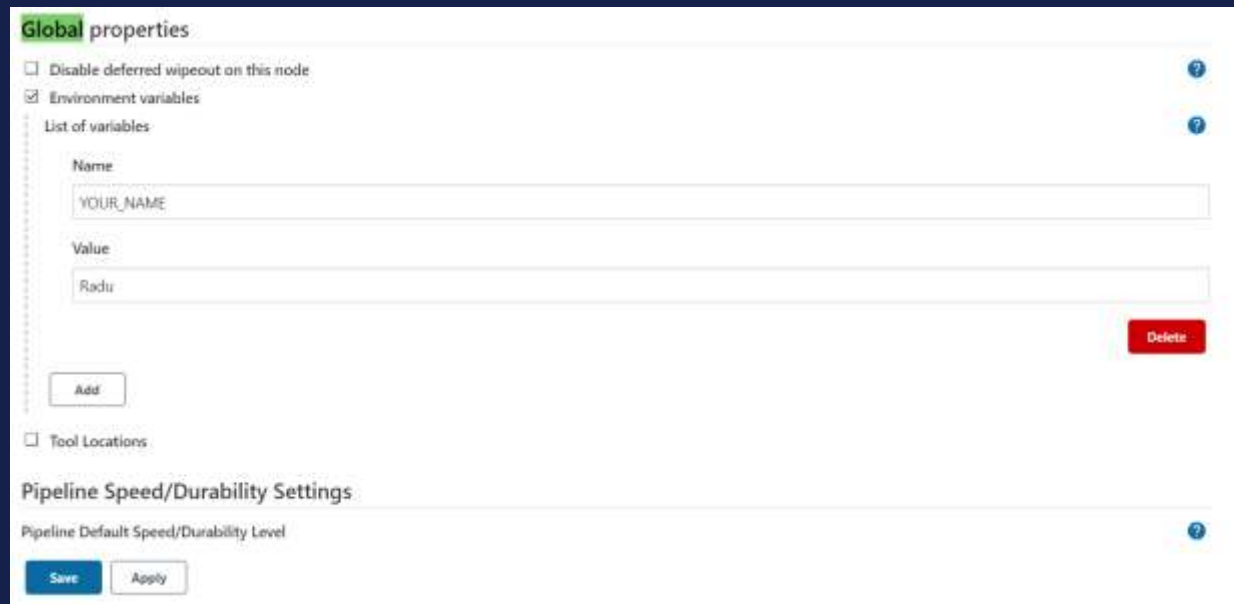
Il putem salva ca: **arhiva_\${JOB_NAME}_\${BUILD_ID}**

Și rezultatul va fi: **arhiva_mytestjob_7**

Variabile globale custom

Manage jenkins -> Configure System -> Global properties -> Environment variables

Atunci când acea variabilă va fi folosită în cadrul unui job, va avea valoarea dată de noi în configurație. Nu uitați să dați click pe **save** pentru a salva valoarea variabilei .



The screenshot shows the 'Global properties' configuration page in Jenkins. The 'Environment variables' section is active, showing a table with one variable: 'YOUR_NAME' with the value 'Radu'. There are 'Add' and 'Delete' buttons. Below this, there are checkboxes for 'Disable deferred wipeout on this node' and 'Tool Locations'. At the bottom, there are 'Save' and 'Apply' buttons.

Name	Value
YOUR_NAME	Radu

Va putea fi utilizat ca și în cazul variabilelor build in cu `${YOUR_NAME}`

Job-uri de jenkins programate

Asemănător cu linux, putem avea job-uri care să se execute în anumite intervale de timp, la fel ca în structura cron .

Build Triggers -> Build periodically -> de exemplu * * * * * va executa job-ul în fiecare minut .

1 * * * * -> va executa job-ul o dată pe oră

Acest lucru poate fi foarte util în cazul în care vrem să executăm job-ul la o perioadă fixă de timp. De exemplu, în cadrul unui proiect dorim să executăm un script la sfârșitul fiecărei zile pentru a face un backup sau să verificăm anumiți parametri pentru a face o analiză ulterioară .

Schimbare jenkins home

Exista cazuri in care dorim sa schimbam directorul de home default din jenkins.
Acest lucru se poate face de [aici](#)

Useri în jenkins

Configure Global Security -> Jenkins' own user database -> Allow users to sign up
Oricine are acces la jenkins își poate crea un nou user .

Dupa ce ai bifat allow user to signup, accesează în jenkins (pe modul incognito) și dă restart. După acest pas ar trebui sa iti poti crea un user.

Jenkins se poate restarta atat cu `systemctl restart jenkins`, dar și direct din browser cu `http://jenkins_ip:8080/restart`

Useri în Jenkins

Pentru a adauga anumite restricții pentru noul user e necesar să se instaleze plugin-ul **Role-based Authorization Strategy**.

Configure Global Security -> Authorization -> Role-Based Strategy -> De aici putem configura ca un user nou să se supună restricțiilor aferente (adică să aibă atât acces cât îi conferă rolul pe care îl vom atribui pe acel user).

Useri în jenkins

Manage -> Manage and assign roles -> Manage Roles

De aici se poate crea un role, care de exemplu să aibă doar drepturi de read.

Manage -> manage and assign roles -> Assign Roles -> User/group to add

Rolul creat se va atribui pentru un anumit user

De asemenea, se poate crea un rol care da access sau nu doar pe anumite job-uri. De exemplu sa ai drepturi să rulez toate job-urile care încep cu QA-TEST-* pentru un anumit user și să nu ai drepturi pe alte job-uri cu alte nume, de exemplu PRODUCTION-* .

In cazul in care ne pierdem parola de admin de jenkins, dar avem access la fisierul de configuratie, ne putem recupera parola , modificând temporar acel [fisier](#).

Jenkins plugins

Plugin-urile au rolul de a îmbunătăți funcționalitățile Jenkins și de a ajuta la integrarea lui cu diferite alte tool-uri sau servicii de cloud

Exemple: Amazon EC2 Plugin, Pipeline Plugin, Maven Plugin, Slack Notification Plugin , Docker plugin, Git plugin

Slack plugin

Putem configura ca de fiecare dată când avem un build reușit să primim o notificare pe slack
Acest lucru se poate face urmând pașii de [aici](#)

Un tutorial complet cu toți pașii se poate regăsi și [aici](#)



Email plugin

Se poate configura astfel încât dacă avem un job reușit (sau chiar unul eșuat) să primim alerte . Putem primi aceste alerte pe aplicații precum **slack**, **teams** sau chiar pe **email**.

Pentru ultimul caz este necesar ca **mailer plugin** să fie instalat. Pentru asta este necesar să configurăm **smtp gmail settings** .

Manage jenkins -> Configure system -> E-mail Notification



Email plugin

Default user e-mail suffix ?

☒ Use SMTP Authentication ?

User Name

Password

[Change Password](#)

☒ Use SSL ?

☐ Use TLS

SMTP Port ?

Reply-To Address

Charset:

☒ Test configuration by sending test e-mail

[Test e-mail recipient](#)

[Save](#) [Apply](#)

Email plugin

Configurare

SMTP server: smtp.gmail.com

Use SMTP Authentication -> adauga email si parola

Use ssl -> yes

SMTP PORT -> 465

Test configuration by sending test e-mail -> Test configuration

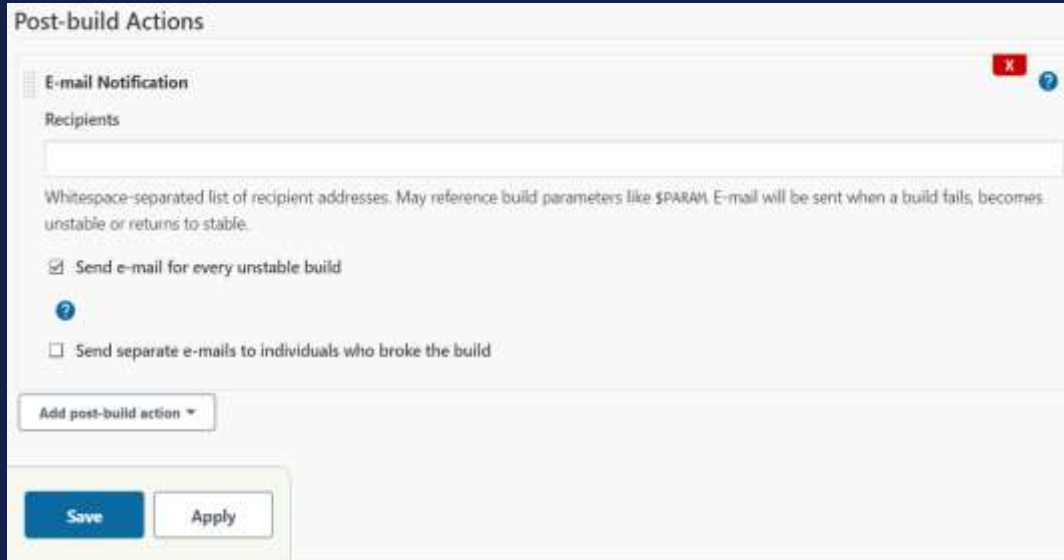
În caz că apare o eroare la trimiterea email-ului de test este recomandat să bifăm ambele setări

- <https://www.google.com/settings/security/lesssecureapps> and Turn On
- <https://accounts.google.com/DisplayUnlockCaptcha>

Email plugin

Nu faceți acest test de pe mail-ul personal, este recomandat să creați un cont de gmail, unde să faceți această verificare .

Când se va executa un build care nu funcționează, veți primi o notificare pe email în acest caz . De asemenea, vom primi și când job-ul va rula din nou fără erori .



The screenshot shows the 'Post-build Actions' configuration interface. The 'E-mail Notification' plugin is selected, indicated by a red 'X' icon. Below the plugin name is a 'Recipients' text input field. A descriptive text below the field states: 'Whitespace-separated list of recipient addresses. May reference build parameters like \$PARAM. E-mail will be sent when a build fails, becomes unstable or returns to stable.' There are two checkboxes: 'Send e-mail for every unstable build' (checked) and 'Send separate e-mails to individuals who broke the build' (unchecked). A blue question mark icon is next to the first checkbox. At the bottom left is a button labeled 'Add post-build action'. At the bottom right are two buttons: 'Save' and 'Apply'.

Git webhook

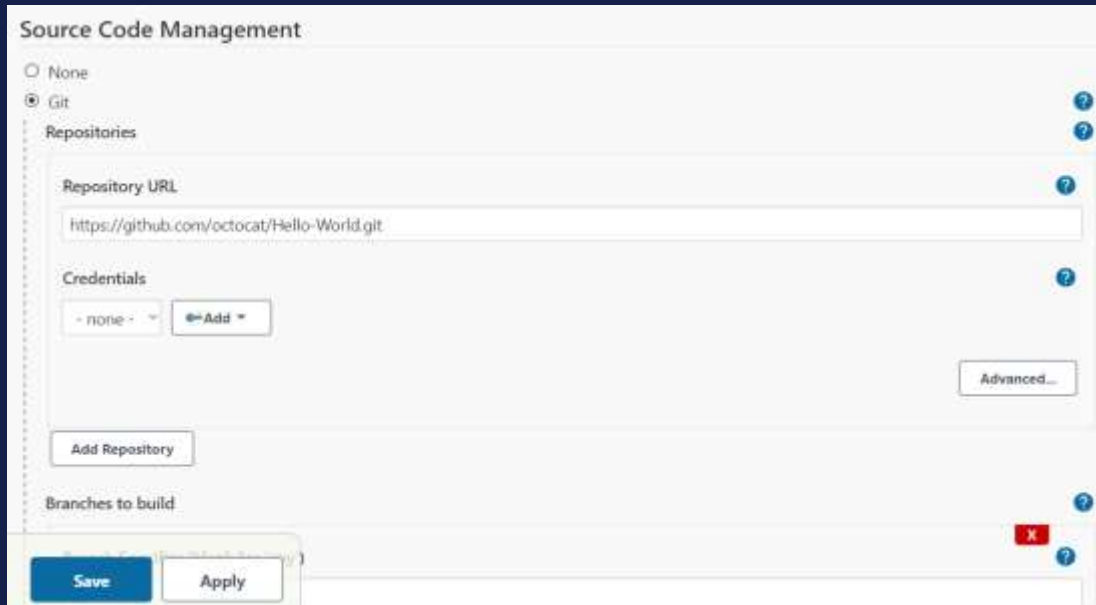
Putem configure git sa trimita catre jenkins o notificare atunci cand pe git se intampla un PR, un push sau un eveniment anume ales de noi. Comunicarea intre git si jenkins se va face prin internet si pentru ca serverul de git ca trimite mesajul catre jenkins, serverul de jenkins trebuie sa fie pe un IP public. Putem folosi un tool precum ngrok pentru a expune temporar serverul de jenkins in internet si a ne folosi de endpoint-ul lui

Jenkins și Git

Pentru asta este nevoie să se instaleze plugin-ul corespunzător: **Git plugin**

Se poate selecta din configurația job-ului, de la **Source Code Management** repo-ul pe care ni-l dorim .

Putem căuta un repo de git public pentru început și copia. După ce rulăm job-ul tot ce era în repo-ul de git, pe branch-ul ales de noi, va fi copiat în **"home jenkins"** adică în cazul de față **"/var/lib/jenkins/"** , în **"workspace"** și se va crea un folder cu numele job-ului executat, în cazul de față 'gitjob'
/var/lib/jenkins/workspace/gitjob .



The screenshot shows the 'Source Code Management' configuration page in Jenkins. The 'Git' option is selected under 'Source Code Management'. Under 'Repositories', the 'Repository URL' is set to 'https://github.com/octocat/Hello-World.git'. The 'Credentials' dropdown is set to 'none' with an 'Add' button next to it. There is an 'Advanced...' button to the right. At the bottom left, there is an 'Add Repository' button. At the bottom, there are 'Save' and 'Apply' buttons. The 'Branches to build' section is partially visible at the bottom.

Credențiale secrete

Pe lângă variabile clear-text putem crea și variabile secrete. Acestea se definesc cu **Dashboard -> Manage Credentials -> System Global credentials (unrestricted)** -> și de exemplu adăugăm o variabilă cu numele parola și o valoare secretă .

Add credentials -> Secret text -> (aici setăm numele variabilei și valoarea)

În shell, prin jenkins vom scrie următoarea comandă:

```
echo "variabila secreta este: $mypass"
```

```
echo "variabila secreta este: $mypass" > /tmp/secret
```

Credențiale secrete

Build Environment

☒ Delete workspace before build starts

Advanced...

☒ Use secret text(s) or file(s) ?

Bindings

Secret text X ?


Variable ?

mypass

Credentials ?

☒ Specific credentials ☐ Parameter expression

parola desc ▼ Add ▼



IT School
FROM ZERO TO HERO!

Ce este un agent jenkins ?

Un agent Jenkins este un executabil, care locuiește pe un nod (sau server), care are ca scop să execute un job .

De exemplu, până acum am folosit ca agent chiar server-ul unde este instalat jenkins. Sa îi spunem acestuia serverul A .

Totuși, se poate configura altă mașină (serverul B) pe care o conectăm la serverul A și de pe serverul A putem rula job-uri pe serverul B.

Astfel, serverul A va fi doar cel care distribuie treaba și serverul B ce face toată procesarea .

Se poate configura chiar și un container de docker care să aibă rolul de agent și care va avea mereu aceeași stare atunci când va fi “up and running” ceea ce ar elimina eventualele erori produse de configurații mai vechi pe un server ce i-ar schimba starea inițială .







Configurarea unui agent

Manage Jenkins -> Manage Nodes and Clouds -> New node -> Give an "Remote root directory" and Name -> Launch method via ssh

Selecteaza IP-ul serverului, si perechea de credentiale pe care o creezi anterior in jenkins

Creaza o pereche de crentiale cu username si parola.

Manage nodes and clouds Refresh status

S	Name	Architecture	Clock Difference	Free Disk Space	Free Swap Space	Free Temp Space	Response Time
	Built-In Node	Linux (amd64)	In sync	1.90 GB	923.21 MB	1.90 GB	0ms 
	server1	Linux (amd64)	In sync	5.23 GB	915.91 MB	5.23 GB	645ms 
Data obtained		4.9 sec	4.8 sec	4.6 sec	4.5 sec	4.6 sec	4.6 sec

Crerea unui build de maven

Am ales un repo simplu de git ([link](#)) pe care îl configurăm prin git scp într-un job de jenkins .

Codul în care este scrisă mini aplicația este java. Pentru a putea să creăm build-ul este necesar să avem maven instalat pe serverul local. Cream un simplu job **Build-demo** ce va avea ca sursa acel repo

Pași de instalare maven:

```
sudo apt update  
sudo apt install maven  
mvn -version
```

Pentru a face build-ul manual vom merge in workspace -> Build-demo și rulăm:
mvn package



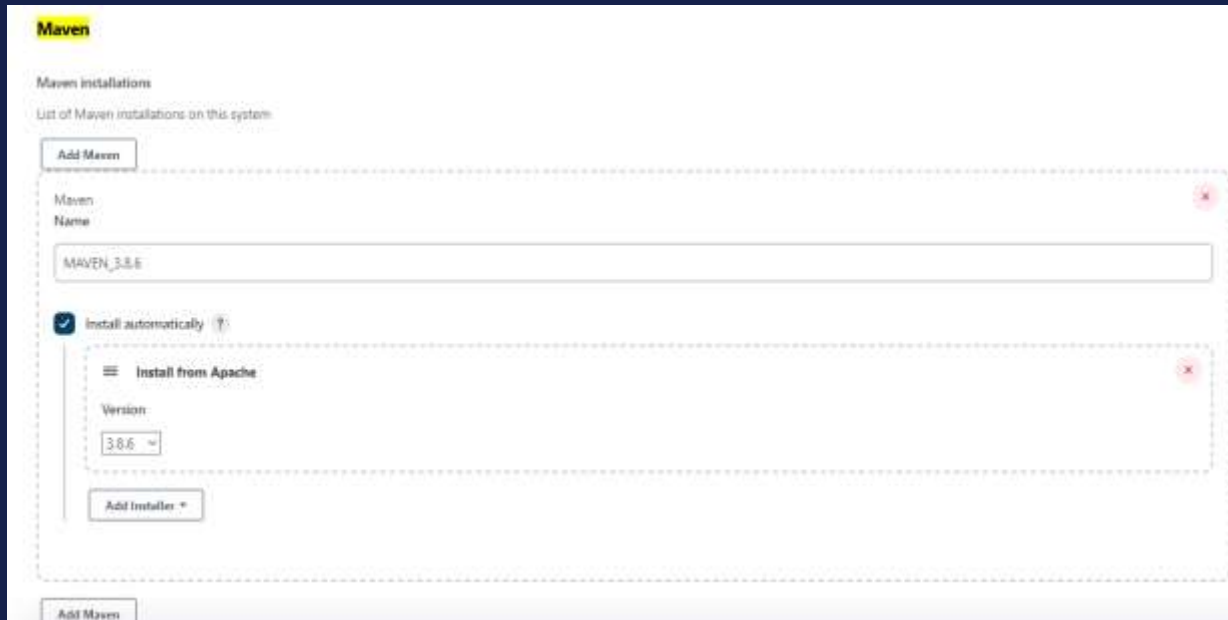
Crerea unui build de maven

Fișierul **.war** creat va putea fi găsit aici .
./target/hello-world-maven.war

Totuși se poate face asta automat direct din jenkins, fără a fi necesar să rulăm comanda manual în linux .

Crerea unui build de maven

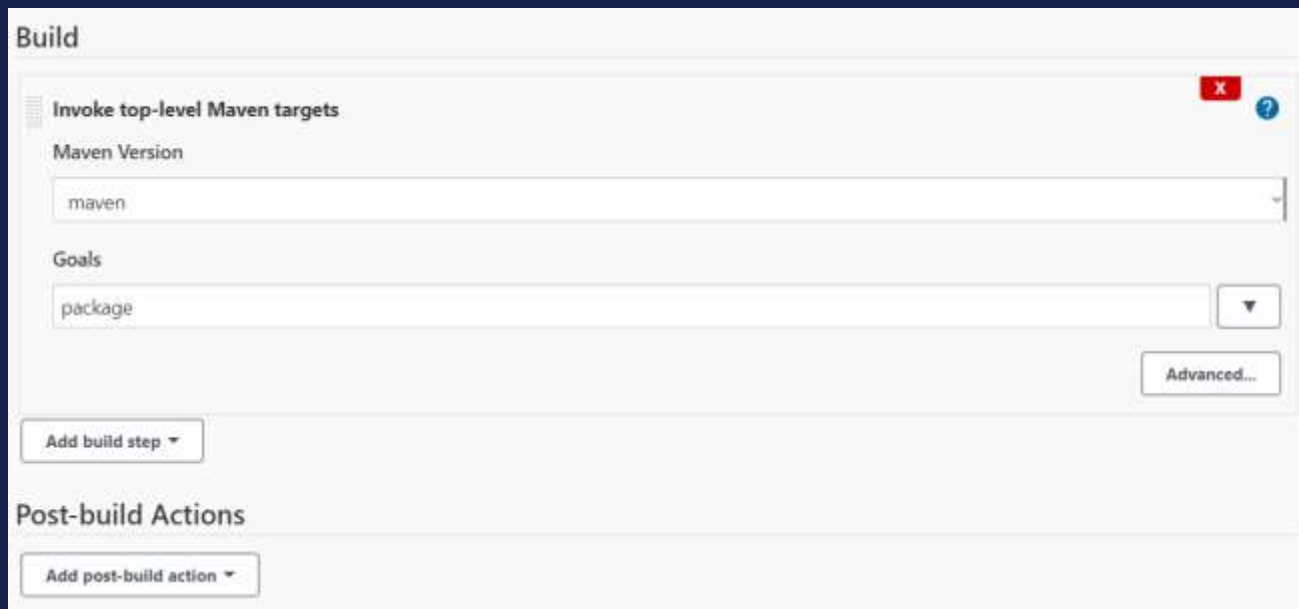
Varianta a doua este să facem asta printr-un plugin de **maven integration**. Se instalează acest plugin. Apoi de la global configuration tool alegem o versiune de maven și un nume.



The screenshot shows the 'Maven' configuration window in an IDE. It has a title bar with the 'Maven' logo. Below the title bar, it says 'Maven installations' and 'List of Maven installations on this system:'. There is an 'Add Maven' button at the top left. The main area is a dashed box containing a 'Maven' section with a 'Name' field set to 'MAVEN_3.8.6'. Below this is a checked 'Install automatically' checkbox. Underneath is an 'Install from Apache' section with a 'Version' dropdown set to '3.8.6' and an 'Add Installer' button. There is also an 'Add Maven' button at the bottom left.

Crerea unui build de maven

După instalarea plugin-ului, se poate vedea în configurația job-ului că apare, la build **Invoke top-level maven targets** .



The screenshot shows the 'Build' configuration section of a Jenkins job. The main heading is 'Build'. Below it, there is a section titled 'Invoke top-level Maven targets' with a red 'X' icon and a help icon. Under this section, there are two input fields: 'Maven Version' with the value 'maven' and 'Goals' with the value 'package'. There is an 'Advanced...' button to the right of the 'Goals' field. Below the 'Invoke top-level Maven targets' section, there is a button 'Add build step ▾'. At the bottom of the configuration, there is a section titled 'Post-build Actions' with a button 'Add post-build action ▾'.

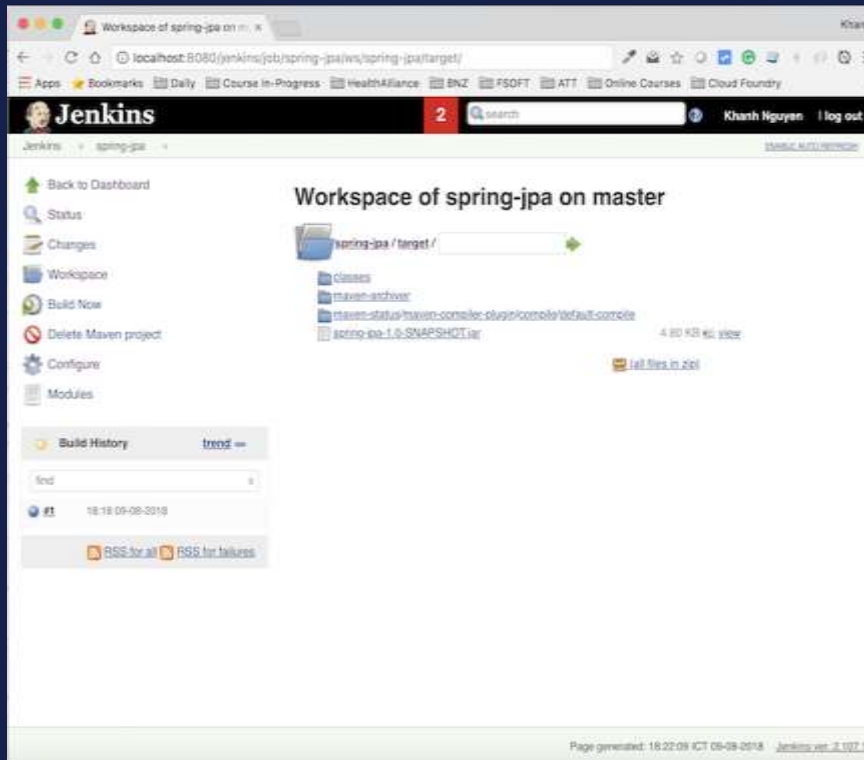
Ce este un artefact ?

Un artefact în Jenkins este un artefact este un fișier imuabil, generat în timpul unei rulări Build sau Pipeline în Jenkins. Aceste artefacte sunt apoi arhivate de Jenkins pentru utilizare ulterioară . Pentru ca alt build sa aiba access la acel artefact e nevoie sa bifam **Permission to Copy Artifact** dar si sa configuram **Post-build Actions** cu **Archive the artifacts**

Si sa specificam artefactul (ex `**/*.war`)

Ce este un artefact ?

Un artefact în Jenkins este un artefact este un fișier imuabil, generat în timpul unei rulări Build sau Pipeline în Jenkins. Aceste artefacte sunt apoi arhivate de Jenkins pentru utilizare ulterioară . Pentru ca alt build sa aiba access la acel artefact e nevoie sa bifam **Permission to Copy Artifact**



Publish the artifact

Post-build Actions -> archive the artifacts

Files to archive -> **/*.war

Pentru a avea drepturi depline pe userul jenkins putem adăuga în sudoers:
jenkins ALL=(ALL) NOPASSWD: ALL

Nu este recomandat, dar se poate face doar pentru testare în cazul în care avem permission denied în anumite locații de pe server și vrem să le rezolvăm doar pentru scopul acestui demo.

Pe job-ul de build e necesar să bifeăm și Permission to Copy Artifact pentru a da voia altor job-uri să copieze artifact-ul creat



Crearea unui release cu docker

Pentru a nu fi nevoie sa instalam pachete pe serverul de jenkins, ne putem folosi de un docker care are deja tomcat instalat si sa il utilizam doar pentru a vedea si citi continutul acelui fisier .war

Ne vom folosi de acest [repo](#)

Si vom selecta **Copy artifacts from another project** pentru a copia artefactul creat anterior

Artifacts to copy -> target/*.war

Target directory -> \${WORKSPACE}

Crearea unui release cu docker

```
docker stop mymavenapp  
docker rm mymavenapp  
docker rmi -f maven  
docker build -t "maven" .  
docker run -d -p 9998:8080 --name mymavenapp maven
```

Si vom accesa aplicatia tomcat astfel:

http://ADRESA_IP:9998/hello-world-maven



Din nou, ca să nu facem pașii aceștia manual. **Arhive artifact** **/target/*.jar sau **/*.war**

Apoi facem un job de release.

Pentru a realiza acest lucru avem nevoie de plugin-ul **Copy Artifact** .

Se poate configura job-ul de **maven_build** astfel încât dacă el se termină de executat fără vreo eroare, să fie un trigger pentru build-ul de release .

Build

☐ Copy artifacts from another project

Project name [?]

maven_build

Which build [?]

Latest successful build

☐ Stable build only

Artifacts to copy [?]

target/*.war

Artifacts not to copy [?]

Target directory [?]

\${WORKSPACE}

Parameter filters [?]

☐ Flatten directories ☐ Optional ☒ Fingerprint Artifacts [?]

Advanced...

Crearea unui release cu instalare tomcat

Până acum am lucrat la partea de CI (**continuous integration**) . Acest artefact cu extensia **.jar** poate fi folosit pentru a realiza un release .

Pentru acest lucru este necesar să instalăm [tomcat](#) . Pași de instalare:

```
sudo apt update  
sudo apt install tomcat9 tomcat9-admin  
sudo systemctl enable tomcat9  
systemctl status tomcat9
```

Alte variante ar fi [aici](#) si se poate descarca de [aici](#)

Tomcat rulează pe portul 8080. Pentru acest curs, vom configura tomcat să ruleze pe portul 9999 pentru a nu avea un conflict de porturi cu serverul de jenkins .



Jenkins și docker

Să presupunem că în repo-ul de git acum un **Dockerfile**. Putem crea o imagine pe baza acelui dockerfile direct din jenkins .

```
docker build -t myimage:${BUILD_ID} .
```

Build:

```
docker run -it --rm --name my-maven-project -v "$(pwd)":/usr/src/mymaven -w /usr/src/mymaven maven:3.3-jdk-8 mvn package
```

Release:

```
docker run -it -v "$(pwd)":/usr/local/tomcat/webapps -p 9999:8080 tomcat:9.0
```



Jenkins și docker

Pentru a nu mai instala nimic pe serverul de jenkins, ne putem folosi de un container de docker, unde vom putea face un build direct pe acel container și să extragem doar artefactul .

Build: `docker run -i --rm --name my-maven-project -v "$(pwd)":/usr/src/mymaven -w /usr/src/mymaven maven:3.3-jdk-8 mvn package`

Apoi folosim **archive the artifact**: `**/*.war`
+ build other projects (release job)

Și va fi nevoie de **Copy artifact plugin**



Jenkins și docker

Release:

Copy artifacts from another project, **artifacts to copy**: `**/*.war`

Apoi rulăm un container de tomcat pentru a pune face un release cu acel artifact.

Se poate realiza cu comenzile de shell:

```
cd target && mv hello-world-war-1.0.0.war webapp.war && docker run -d -v  
"$(pwd)":/usr/local/tomcat/webapps -p 9999:8080 --name tomcatcontainer-  
${BUILD_ID} tomcat:9.0 tail -f /dev/null
```

```
docker exec tomcatcontainer-${BUILD_ID} ./bin/catalina.sh start
```



Pipeline script

Ce este un pipeline ?

Un pipeline reprezintă un set de instrucțiuni sub forma unui cod pentru a asigura un intreg flow de CI/CD . Cu un pipeline se poate testa, face un build sau livra aplicația . Limbajul de programare în care se va scrie acest script este Groovy .

Exemplu de script:

```
pipeline {  
  agent any  
  stages {  
    stage('Build') {  
      steps {  
        sh 'echo "hello world"'  
      }  
    }  
  }  
}
```

Pipeline script

Pipeline

Definition

Pipeline script

Script

1

try sample Pipeline...

Pipeline script

Pentru acest pipeline putem avea mai multe stage-uri, unul de build, altul de testare sau deploy în funcție de ce avem nevoie .

```
pipeline {  
  agent any  
  stages {  
    stage('Build') {  
      steps {  
        echo 'Building..'  
      }  
    }  
    stage('Test') {  
      steps {  
        echo 'Testing..'  
      }  
    }  
    stage('Deploy') {  
      steps {  
        echo 'Deploying....'  
      }  
    }  
  }  
}
```



Pipeline script

De recomandat este să nu scriem codul groovy direct în jenkins. De aceea există varianta să se preia definiția script-ului direct din SCM (Source code management, în cazul de față din git). Tot ce trebuie făcut este ca în repo-ul de git să avem un fișier Jenkinsfile cu acel cod .

Parametrii în Jenkinsfile

Tot prin Jenkins file poți defini parametri și să fie folosiți exact ca în cazul în care îi se creează manual din interfața grafică .

```
pipeline {  
  agent any  
  parameters {  
    string(name: '', defaultValue: 'Hello', description: 'Short description')  
  }  
  stages {  
    stage('Example') {  
      steps {  
        echo "${params.Greeting} World!"  
      }  
    }  
  }  
}
```

Cron în jenkinsfile

```
pipeline {  
  agent any  
  triggers { cron('* * * * *') }  
  stages {  
    stage('Example') {  
      steps {  
        echo 'Hello World' }  
      }  
    }  
  }  
}
```

Jenkinsfile și git

```
pipeline {
  agent any
  stages {
    stage('Test') {
      steps {
        git branch: 'main', credentialsId: 'gitaccount', url: 'git_repo_link' }
      }
    }
  }
}
```

'**gitaccount**' trebuie creat ca username and password în credențialele din jenkins . Atunci când se creează aceste credențiale, este necesar ca în loc de id-ul de git și parolă, să punem id-ul de git și un token generat din setările contului din **Developer settings -> Personal access token** . Este necesar ca acel token să aibă un minim de drepturi necesare pentru a face operațiunile dorite, de exemplu dacă pentru acest token nu avem drepturi de push, evident că nu vom putea face asta cu acest tip de autentificare .

Jenkinsfile build și publish artifact

Se poate crea un build , dar și publica acel artifact direct din codul de Jenkinsfile .

```
pipeline {  
    agent any  
    stages {  
        stage('scm') { steps { git branch: 'master', credentialsId: 'gitaccount', url:  
'https://github.com/gxg513/ITSHOOL4TOMCAT.git' }}  
        stage('Build') { steps { sh 'mvn package' } }  
        stage('archive the artifacts') { steps { archive '**/*.war' } }  
    }  
}
```

Exemplu de Jenkinsfile

În acest caz definim o variabilă cu environment, stergem directorul vechi **deleteDir()** , se descarcă repo-ul de git și se creează build-ul .war .

```
pipeline {
  agent any
  environment {
    DOCKER_IMAGE_NAME = "gxc513/webapp.war"  }
  stages {
    stage('test-ls') {
      steps {
        deleteDir()
        sh 'pwd'
        sh 'ls -lah'      }    }
    stage('scm') {
      steps { git branch: 'master', credentialsId: 'gitaccount', url:
'https://github.com/gxc513/ITSHOOL4TOMCAT.git'  }}
    stage('Build') {
      steps { sh 'mvn package' } }
  }
}
```



Sfârșit

