

ბიზნესისა და ტექნოლოგიების უნივერსიტეტი

# ვერსიონირება და უნყვეტი ინტეგრაცია

თემა მეექვსე და მეშვიდე:  
**Jenkins Job-ები, ტრიგერები და  
დაკავშირება**

## Jenkins Job-ები და ტრიგერები

Jenkins Job არის Jenkins-ის ცენტრალური კონცეფცია და მისი ფუნქციონირების გული. მარტივად რომ ვთქვათ, Job-ი არის ავტომატიზებული პროცესის ან დავალების აღწერა, რომელსაც Jenkins-ი ასრულებს. ეს პროცესი შეიძლება იყოს ნებისმიერი რამ: პროგრამული კოდის კომპილაცია, ტესტების გაშვება, აპლიკაციის სერვერზე განთავსება, მონაცემთა ბაზის განახლება ან თუნდაც სხვა სისტემებისთვის შეტყობინების გაგზავნა. თითოეული Job-ი წარმოადგენს CI/CD (Continuous Integration/Continuous Deployment) პროცესის ერთ კონკრეტულ, შესასრულებელ ნაბიჯს.

Job-ების გაშვება შესაძლებელია როგორც ხელით ასევე ტრიგერების მეშვეობით. ტრიგერი განსაზღვრავს, თუ როდის უნდა დაიწყოს Job-მა მუშაობა.

- **Trigger builds remotely (დისტანციური გაშვება URL-ით)** - ეს ტრიგერი საშუალებას გვაძლევს, Job-ი გავუშვათ უნიკალურ ბმულზე გადასვლით, რომელსაც ჩვენ თვითონ მოვიფიქრებთ. ამისთვის იქმნება უნიკალური ავთენტიფიკაციის ტოკენი, სტრიქონი რომელიც ჯენკინსის Job-ის ბოლოში მიენერება, მაგალითად `localhost:8080/job/test_job/build?token=tokenis_saxeli`

**Build Triggers**

☒ Trigger builds remotely (e.g., from scripts) ?

Authentication Token

Use the following URL to trigger build remotely:

JENKINS\_URL/job/jenkins-github-webhook-example/build?  
token=TOKEN\_NAME or /buildWithParameters?  
token=TOKEN\_NAME

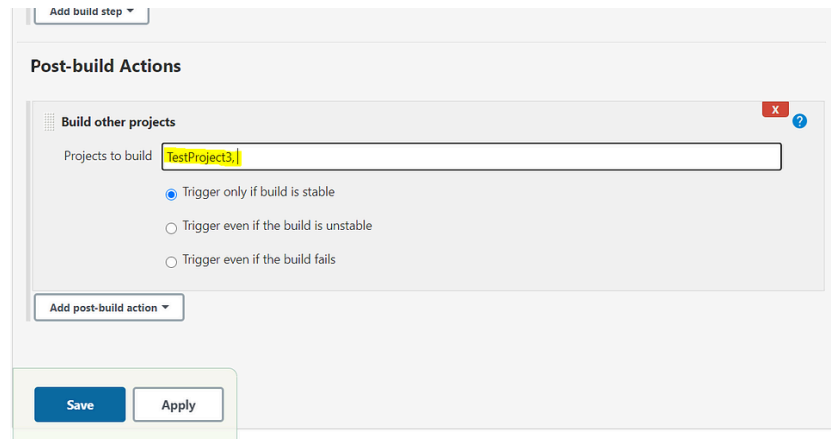
Optionally append &cause=Cause+Text to provide text that  
will be included in the recorded build cause.

☐ Build after other projects are built ?

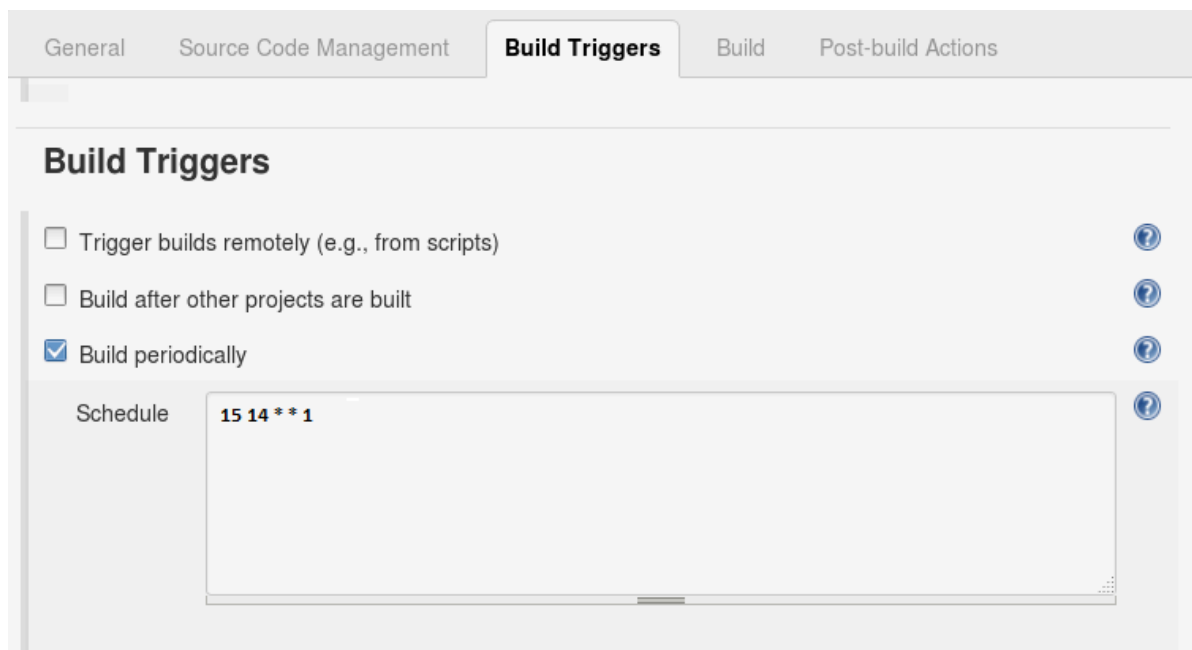
☐ Build periodically ?

- **Build after other projects are built (დამოკიდებული Job-ები)** - ეს ტრიგერი ერთ Job-ს მეორის დასრულებაზე დამოკიდებულს ხდის. Job B-ს კონფიგურაციაში უთითებთ, რომ ის უნდა გაეშვას მას შემდეგ, რაც Job A დაასრულებს მუშაობას. შესაძლებელია დაკონკრეტება, თუ რა შედეგზე უნდა მოხდეს რეაგირება (მაგ. მხოლოდ წარმატებით

დასრულებისას, ან თუნდაც წარუმატებლობისას). აღნიშნული მეთოდით შესაძლებელია Job-ების მუშაობის კომპლექსური სტრუქტურის შექმნა.



- **Build periodically (Job-ის პერიოდული გაშვება)** - ეს ტრიგერი Job-ს წინასწარ განსაზღვრული განრიგით (Cron-ით) უშვებს. Job-ის კონფიგურაციაში, **Schedule** ველში იწერება Cron-ის ფორმატის გამოსახულება, რომელიც განსაზღვრავს გაშვების დროს. Cron Job-ის განრიგი აღიწერება ხუთი ველით: წუთი, საათი თვის რიცხვი, თვე, კვირის დღე. მაგალითად:  
**15 14 \* \* 1** - გაუშვებს Job-ს ყოველ ორშაბათს 14:15 დროს



- **Webhook ტრიგერი** - ტრიგერი, რომელიც ამუშავებს ჯობს GitHub-ზე ან gitlab-ზე კოდის ცვლილებისას (კონკრეტულად git push-ის შემთხვევაში). ამისათვის jenkins-ი უნდა იყოს დაკავშირებული სხვა სისტემაზე მაგალითად github-ზე და Job-ის კონფიგურაციაში უნდა ჩაინეროს რომელ ბრენჩს უნდა აკონტროლებდეს jenkins-ი. აღნიშნული მეთოდის გამოყენებისას, ყოველ ჯერზე როდესაც github-ზე ახალი კომითი დაიფუშება github-ი გადასცემს შეტყობინებას jenkins-ის Webhook URL-ზე, რის შემდეგაც jenkins-ი იწყებს Job-ის შესრულებას.

GitHub

GitHub Servers

GitHub Server

Name

API URL

Credentials - none - Add

Test connection

☒ Manage hooks

GitHub client cache size (MB)

Delete

Add GitHub Server

Override Hook URL

☒ Specify another hook URL for GitHub configuration

Shared secret - none - Add

Additional actions Manage additional GitHub actions

Re-register hooks for all jobs

Jenkins Job-ების რამოდენიმე ტიპი არსებობს:

- **Freestyle Project** - ეს არის Jenkins-ის კლასიკური და ყველაზე მარტივი Job-ის ტიპი. მისი კონფიგურაცია მთლიანად ვებ-ინტერფეისიდან ხდება, სადაც მომხმარებელი გრაფიკულად, ველების შევსებითა და ოფციების მონიშვნით აწყობს მთლიან პროცესს.
- **Pipeline** - ეს არის Job-ის ტიპი რომელიც ეფუძნება Pipeline as Code პრინციპს. Freestyle-ისგან განსხვავებით, აქ მთელი პროცესი აღინერება კოდის სახით სპეციალურ ფაილში, რომელსაც Jenkinsfile ეწოდება. ეს ფაილი ინახება კოდის რეპოზიტორში, თავად პროექტის კოდთან ერთად.
- **Multibranch Pipeline** - ეს არის ერთგვარი Job-ების გენერატორი, მისი მუშაობის პრინციპი შემდეგია. ის აკვირდება github-ის ბრენცებს და თუ სადმე აღმოაჩენს jenkinsfile-ს მისთვის დააგენერირებს job-ს

- **Multi-configuration project** - ეს არის Job-ი რომელიც ძირითადად ტესტირებისას გამოიყენება, ის უშვებს ერთსა და იმავე პროცესს სხვადასხვა პარამეტრებით ან პარამეტრების კომბინაციებით
- **Folder** - ეს არის კონტეინერი, რომელიც სხვა Job-ების ორგანიზებისთვის გამოიყენება, ერთგვარი საქარალდე.
- **Organization Folder** - ეს არის Job-ი რომელიც ასკანერებს დაკავშირებულ github ან gitlab რეპოზიტორიებს და თუ სადმე იპოვის jenkinsfile-ს შექმნის შესაბამის Multibranch Pipeline-ს.

## SSH

Jenkins არის ხელსაწყო რომელიც გამოიყენება ავტომატიზაციისთვის, ცხადია ის უნდა იყოს დაკავშირებული როგორც სხვადასხვა წყაროებზე, მაგალითად github-ზე ან gitlab-ზე, ასევე უნდა იყოს დაკავშირებული სხვადასხვა სერვერებზე, იმისთვის რომ დატვირთვა გადაიტანოს მათზე და **master node** (სერვერი რომელზეც აყენია jenkins-ი) არ დატვირთოს. ეგეთ სერვერებს რომლებიც გამოიყენება Job-ების დასამუშავებლად ეძახიან **Jenkins client** სერვერებს ან უბრალოდ **slave**-ებს. გარდა ამისა ჯენკინსი როგორც წესი დაკავშირებულია სხვადასხვა ასერვერებზე რომლებზეც იწყოება და იტესტება პროგრამები და ყველა ზემოთნახსენები კავშირის დასამყარებლად როგორც წესი გამოიყენება SSH.

განვიხილოთ jenkins-ისა და github-ის დაკავშირების მაგალითი. SSH კავშირის დასამყარებლად jenkins-სა და github-ს შორის ჩვენ უნდა ავტვირთოთ SSH public key github-ზე ანალოგიურად როგორც ვტვირთავდით ლოკალური გარემოდან public key-ს github-ზე კავშირის დასამყარებლად. Jenkins-ის მხარეს უნდა დავამატოთ private key Credentials გრაფაში manage jenkins-იდან. კორექტული კავშირის დამყარებისათვის საჭიროა რამოდენიმე plugin-ის დაყენება: **Git Plugin, SSH Credentials Plugin, GitHub Integration Plugin**. მსგავსი პრინციპით მყარდება კავშირი სხვა გარემოებთან თუმცა პლაგინები შერჩეული უნდა იყოს კონკრეტული გარემოსთვის და კავშირის მეთოდისთვის.