where we are using and give full groovy script on $class with detailed

**Copilot said:**

**Copilot**

Great! Let's go step-by-step and build a **complete Jenkins Scripted Pipeline** that uses the $class syntax to **trigger a downstream job** using step([$class: 'BuildTriggerStep']).

**🎯 Goal**

We want to:

* Trigger a downstream job called deploy-job
* Pass parameters to it
* Wait for it to finish
* Use the **$class syntax** (low-level Groovy form)

**✅ Full Scripted Pipeline Example with $class**

node {

    stage('Build') {

        echo 'Building the application...'

        // Simulate build

        sh 'mkdir -p build && echo "jar content" > build/app.jar'

    }

    stage('Trigger Deployment Job') {

        echo 'Triggering downstream deployment job...'

        step([

            $class: 'BuildTriggerStep',

            job: 'deploy-job', // Replace with your actual downstream job name

            wait: true,

            parameters: [

                [$class: 'StringParameterValue', name: 'ENV', value: 'staging'],

                [$class: 'BooleanParameterValue', name: 'AUTO\_DEPLOY', value: true]

            ]

        ])

    }

    stage('Post-Trigger') {

        echo 'Downstream job completed. Continuing pipeline...'

    }

}

**🔍 Detailed Explanation**

| **Part** | **Description** |
| --- | --- |
| node { ... } | Scripted pipeline block that runs on a Jenkins agent |
| stage('Build') | Simulates a build step |
| step([$class: 'BuildTriggerStep', ...]) | Triggers another Jenkins job using low-level plugin class |
| job: 'deploy-job' | Name of the downstream job to trigger |
| wait: true | Waits for the downstream job to finish before continuing |
| parameters: [...] | Passes parameters to the downstream job |
| StringParameterValue | Sends a string parameter (e.g., environment name) |
| BooleanParameterValue | Sends a boolean parameter (e.g., auto-deploy flag) |

**🛠️ How to Use This**

1. Make sure the **downstream job (deploy-job)** exists and accepts the parameters ENV and AUTO\_DEPLOY.
2. Paste this script into a **Pipeline job** in Jenkins.
3. Run the job and observe how it triggers the downstream job.

Would you like a version of this using **Declarative Pipeline syntax** or with **conditional logic** (e.g., only trigger if build succeeds)?

**Copy**

AI-generated content may be incorrect

You said:

🛠️ How to Use This

Make sure the downstream job (deploy-job) exists and accepts the parameters ENV and AUTO\_DEPLOY.

Paste this script into a Pipeline job in Jenkins.

Run the job and observe how it triggers the downstream job. give deply job also with detailed

**Copilot said:**

**Copilot**

Great! Let's now create the **downstream job (deploy-job)** that will be triggered by the upstream pipeline. This job will:

* Accept two parameters: ENV (environment name) and AUTO\_DEPLOY (a boolean flag).
* Simulate a deployment process based on those parameters.

**✅ Step-by-Step: Create the Downstream Job (deploy-job)**

**1. Create a New Pipeline Job**

* Go to Jenkins dashboard.
* Click **“New Item”**.
* Enter name: deploy-job.
* Choose **“Pipeline”** and click **OK**.

**2. Configure Parameters**

* In the job configuration:
  + Check **“This project is parameterized”**.
  + Add:
    - **String Parameter**:
      * Name: ENV
      * Default Value: staging
    - **Boolean Parameter**:
      * Name: AUTO\_DEPLOY
      * Default Value: true

**3. Pipeline Script for deploy-job**

Paste this into the **Pipeline script** section:

pipeline {

    agent any

    parameters {

        string(name: 'ENV', defaultValue: 'staging', description: 'Deployment environment')

        booleanParam(name: 'AUTO\_DEPLOY', defaultValue: true, description: 'Auto deploy flag')

    }

    stages {

        stage('Prepare') {

            steps {

                echo "Preparing to deploy to environment: ${params.ENV}"

                echo "Auto-deploy is set to: ${params.AUTO\_DEPLOY}"

            }

        }

        stage('Deploy') {

            when {

                expression { return params.AUTO\_DEPLOY }

            }

            steps {

                echo "Deploying application to ${params.ENV}..."

                // Simulate deployment

                sh "echo 'Deploying to ${params.ENV} environment...'"

            }

        }

        stage('Skip Deploy') {

            when {

                expression { return !params.AUTO\_DEPLOY }

            }

            steps {

                echo "Auto-deploy is disabled. Skipping deployment."

            }

        }

    }

}

**🔍 What This Does**

* **Reads parameters** passed from the upstream job.
* **Conditionally deploys** based on the AUTO\_DEPLOY flag.
* Uses when blocks to control stage execution.

**✅ Final Setup**

Now, when you run your **upstream job**, it will:

* Trigger deploy-job
* Pass ENV and AUTO\_DEPLOY values
* Wait for it to finish
* Continue its own pipeline