Jenkins

Jenkins is typically run as a standalone application in its own process with the built-in Java servlet container/application server (Jetty).

Downloading and running Jenkins in Docker

- Docker image to use is the **jenkinsci/blueocean** image. This image contains the current Long-Term Support (LTS) release of Jenkins (which is production-ready) bundled with all Blue Ocean plugins and features. This means that you do not need to install the Blue Ocean plugins separately.
- A new **jenkinsci/blueocean** image is published each time a new release of Blue Ocean is published. You can see a list of previously published versions of the **jenkinsci/blueocean** image on the tags page

```
docker run -u root -d -p 8080:8080 -v jenkins-data:/var/jenkins_home -v /var/run/docker.sock:/var/run/docker.sock jenkinsci/blueocean
```

- `brew install jenkins-lts`

Unlocking Jenkins

- Browse to localhost:8080 (or whichever port you configured for Jenkins when installing it) and wait until the Unlock Jenkins page appears.
- From the Jenkins console log output, copy the automatically-generated alphanumeric password (between the 2 sets of asterisks).
- On the Unlock Jenkins page, paste this password into the Administrator password field and click Continue.
- **Install suggested plugins -** to install the recommended set of plugins, which are based on most common use cases.
- **Select plugins to install -** to choose which set of plugins to initially install. When you first access the plugin selection page, the suggested plugins are selected by default.

Creating the first administrator user

- When the Create First Admin User page appears, specify the details for your administrator user in the respective fields and click Save and Finish.
- When the Jenkins is ready page appears, click Start using Jenkins.
- This page may indicate Jenkins is almost ready! instead and if so, click Restart.
- If the page does not automatically refresh after a minute, use your web browser to refresh the page manually.
- If required, log in to Jenkins with the credentials of the user you just created and you are ready to start using Jenkins!

What Is CI

- Continuous Integration is a development approach in which members work on the same project coordinate to integrate their work more frequently.
- The code is integrated into a shared repository, and any integration is tested by automated test cases or sequences to look for an error.
- Each team member is expected to integrate their code at least once a day or more as and when required.

Advantages Of Jenkins CI/CD

- An open community operates Jenkins. They host public meetings each month and seek feedback from customers on the progress of the Jenkins initiative.
- Jenkins also embraces cloud-based design to facilitate cloud-based platforms.
- Jenkins offers over 1500+ extensions in its plugins folder. Through plugins, Jenkins CI/CD is much more efficient and mature in comparison to other tools.
- Jenkins CI/CD offers fantastic support for parallel test execution. This helps accelerate existing test processes by running them in parallel.

Jenkins Pipeline

- Jenkins Pipeline (or simply "Pipeline" with a capital "P") is a suite of plugins which supports implementing and integrating continuous delivery pipelines into Jenkins.
- A continuous delivery pipeline is an automated expression of your process for getting software from version control right through to your users and customers. Every change to your software goes through a complex process on its way to being released.
- This process involves building the software in a reliable and repeatable manner, as well as the progression of the built software (called a "build") through multiple stages of testing and deploymentt

Defining a Pipeline

- Scripted Pipeline is written in Groovy. The relevant bits of Groovy syntax will be introduced as necessary in this document, so while an understanding of Groovy is helpful, it is not required to work with Pipeline.
- A basic Pipeline can be created in either of the following ways:

1) By entering a script directly in the Jenkins web UI.
2) By creating a Jenkinsfile which can be checked into a project's source control repository.

Jenkins Pipeline

- The Jenkins pipeline has an expandable automation system for building basic or complicated ‘template’ distribution pipelines via the Domain-specific language (DSL) used in the pipeline. There are four states of Continuous Delivery in Jenkins pipeline-
- Build
- Deploy
- Test
- Release

Advantages Jenkins Pipeline

- By using Groovy DSL (Domain Specific Language), it models easy to complex pipelines as code.
- The code is stored in the form of a text file called ‘**Jenkinsfile**’ that can be scanned into Source Code Management.
- It supports complex pipelines by adding conditional loops, forks, or joining operations and allowing parallel execution tasks.
- It improves user experience by integrating user feedback into the pipeline.
- It’s resilient in terms of Jenkins’ master unplanned restart.
- It can resume from checkpoints saved.
- It can incorporate multiple additional plugins and add-ins.

Jenkins Pipeline

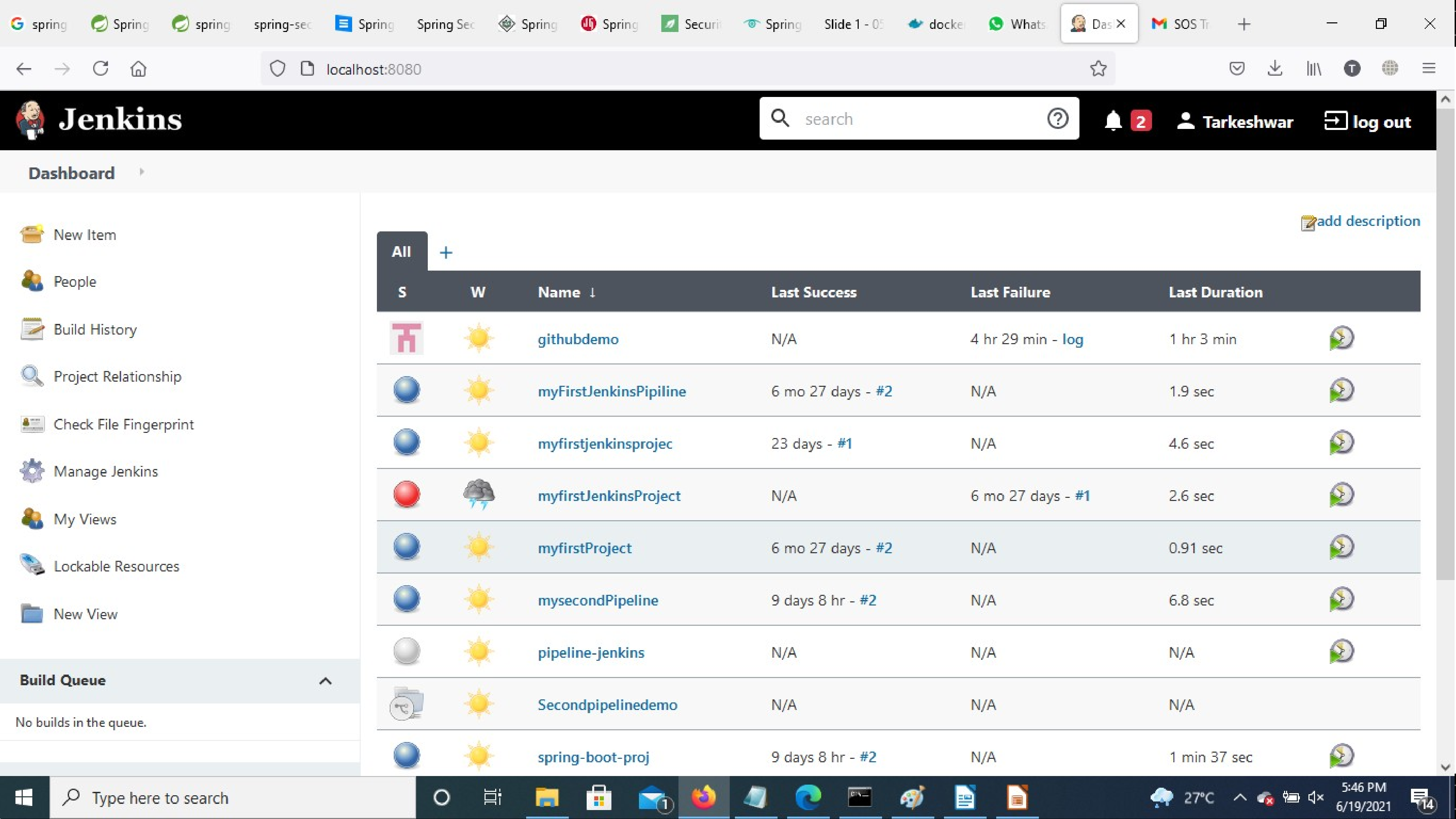
```
// Declarative //
pipeline {
  agent any ①
  stages {
    stage('Build') { ②
        steps { ③
            sh 'make' ④
        }
    }
  stage('Test'){
        steps {
            sh 'make check'
           junit 'reports/**/*.xml'  ⑤
        }
  }
  stage('Deploy') {
        steps {
              sh 'make publish'
        }
    }
  }
}
```

Jenkins Pipeline

```
// Script //
node {
  stage('Build') {
      sh 'make'
    }
  stage('Test') {
      sh 'make check'
      junit 'reports/**/*.xml'
  }
  stage('Deploy') {
      sh 'make publish'
  }
}
```

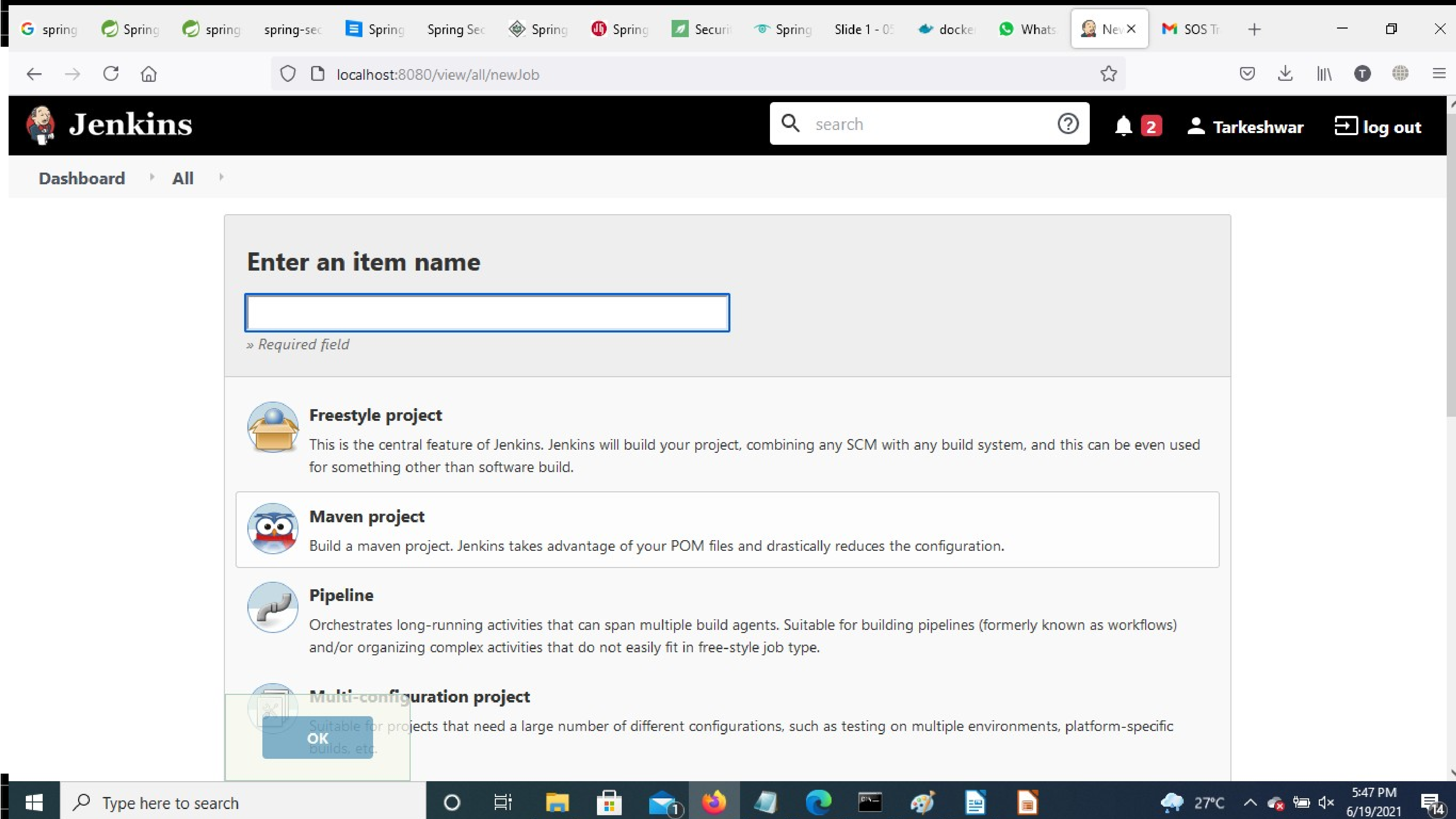
Why Pipeline?

- **Code:** Pipelines are implemented in code and typically checked into source control, giving teams the ability to edit, review, and iterate upon their delivery pipeline.
- **Durable:** Pipelines can survive both planned and unplanned restarts of the Jenkins master.
- **Pausable:** Pipelines can optionally stop and wait for human input or approval before continuing the Pipeline run.
- **Versatile:** Pipelines support complex real-world continuous delivery requirements, including the ability to fork/join, loop, and perform work in parallel.
- **Extensible:** The Pipeline plugin supports custom extensions to its DSL [1: Domain-Specific Language] and multiple options for integration with other plugins.



No builds in the queue.

S	W	Name ↓	Last Success	Last Failure	Last Duration
	☀	githubdemo	N/A	4 hr 29 min - log	1 hr 3 min
●	☀	myFirstJenkinsPipiline	6 mo 27 days - #2	N/A	1.9 sec
●	☀	myfirstjenkinsprojec	23 days - #1	N/A	4.6 sec
🔴		myfirstJenkinsProject	N/A	6 mo 27 days - #1	2.6 sec
●	☀	myfirstProject	6 mo 27 days - #2	N/A	0.91 sec
●	☀	mysecondPipeline	9 days 8 hr - #2	N/A	6.8 sec
○	☀	pipeline-jenkins	N/A	N/A	N/A
	☀	Secondpipelinedemo	N/A	N/A	N/A
●	☀	spring-boot-proj	9 days 8 hr - #2	N/A	1 min 37 sec


localhost:8080/view/all/newJob

Jenkins

Dashboard › All ›

Enter an item name

» 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.

Maven project

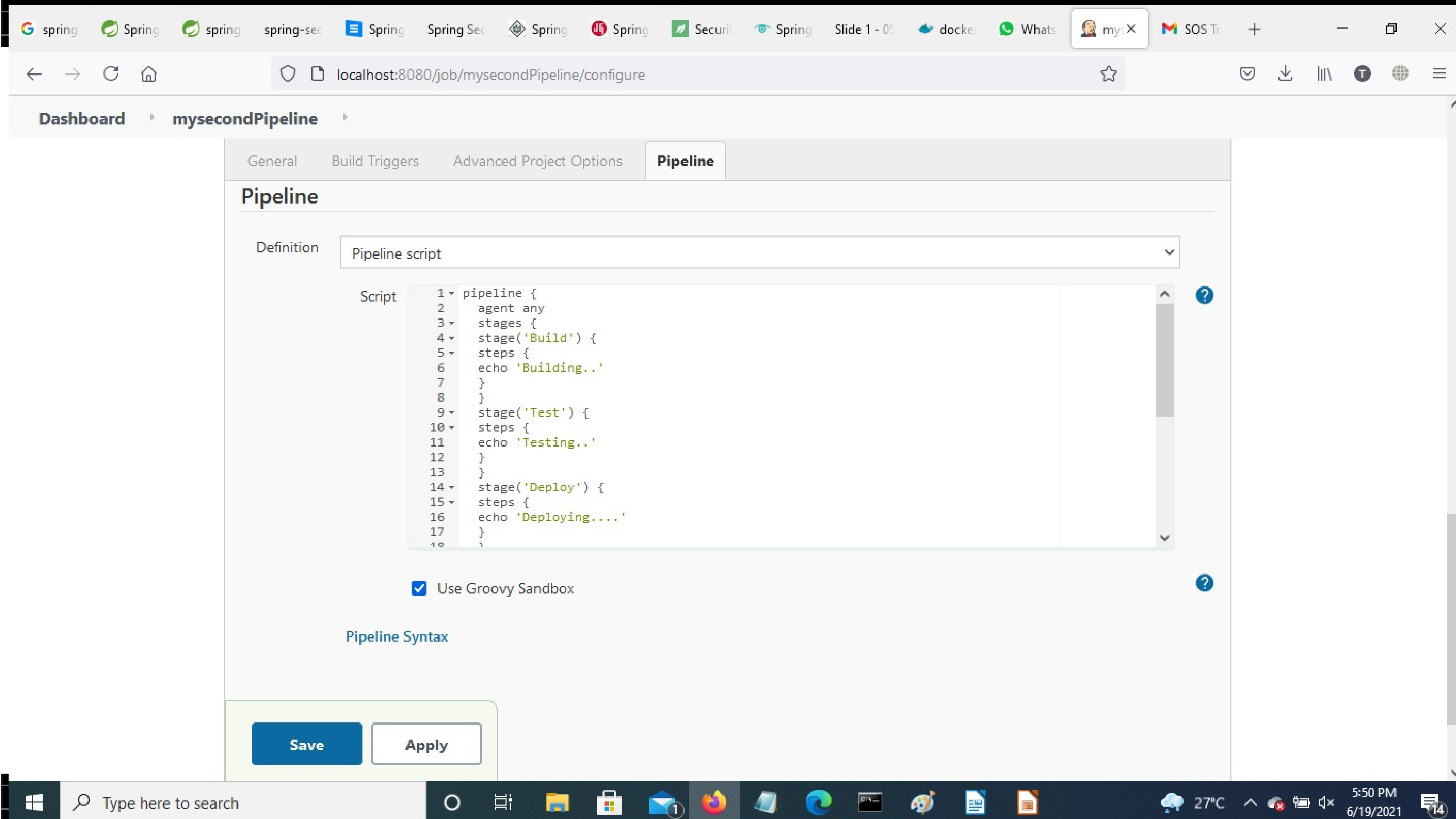
Build a maven project. Jenkins takes advantage of your POM files and drastically reduces the configuration.

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.



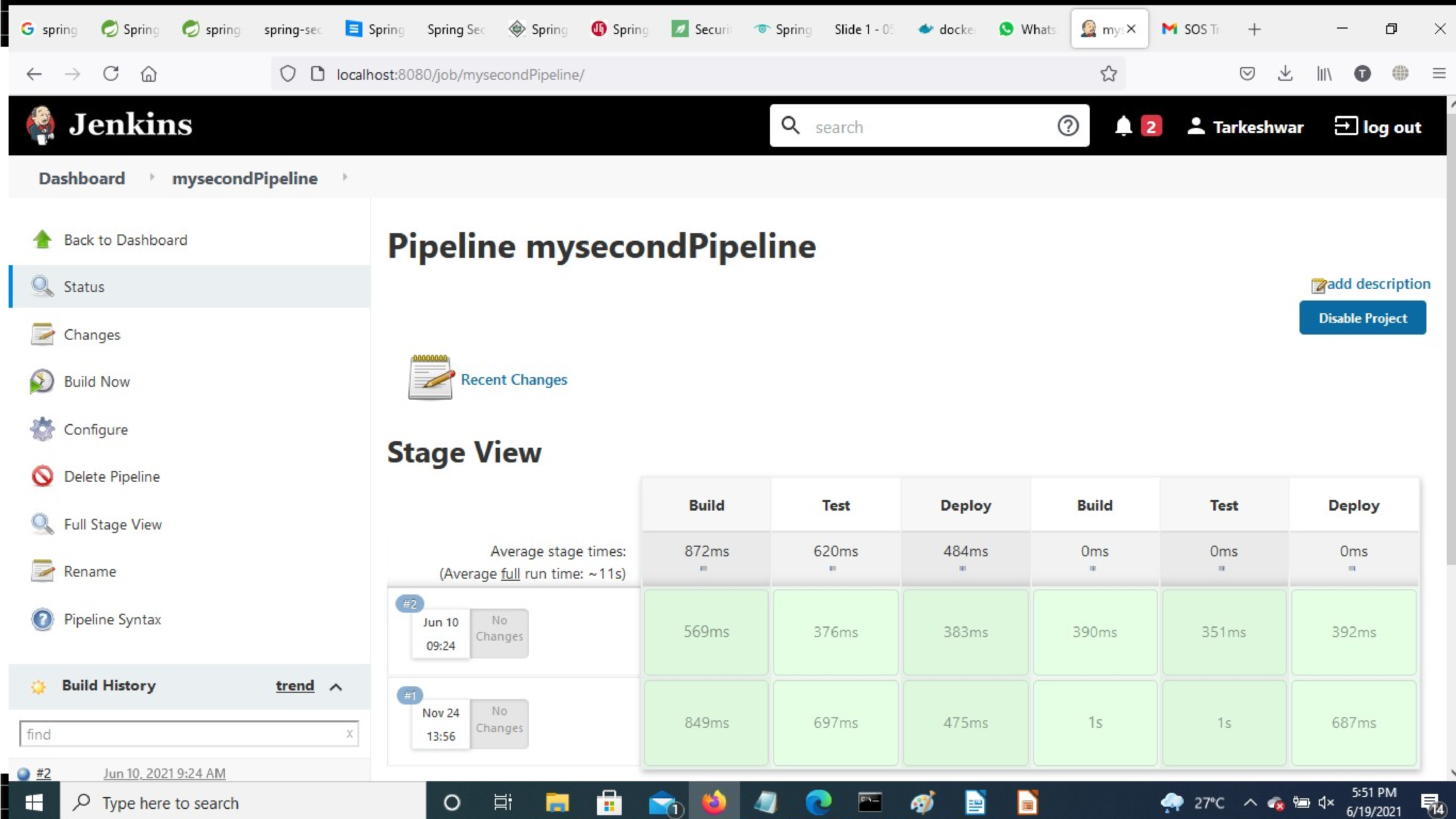
Pipeline

Definition: Pipeline script

Script

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

☑ Use Groovy Sandbox

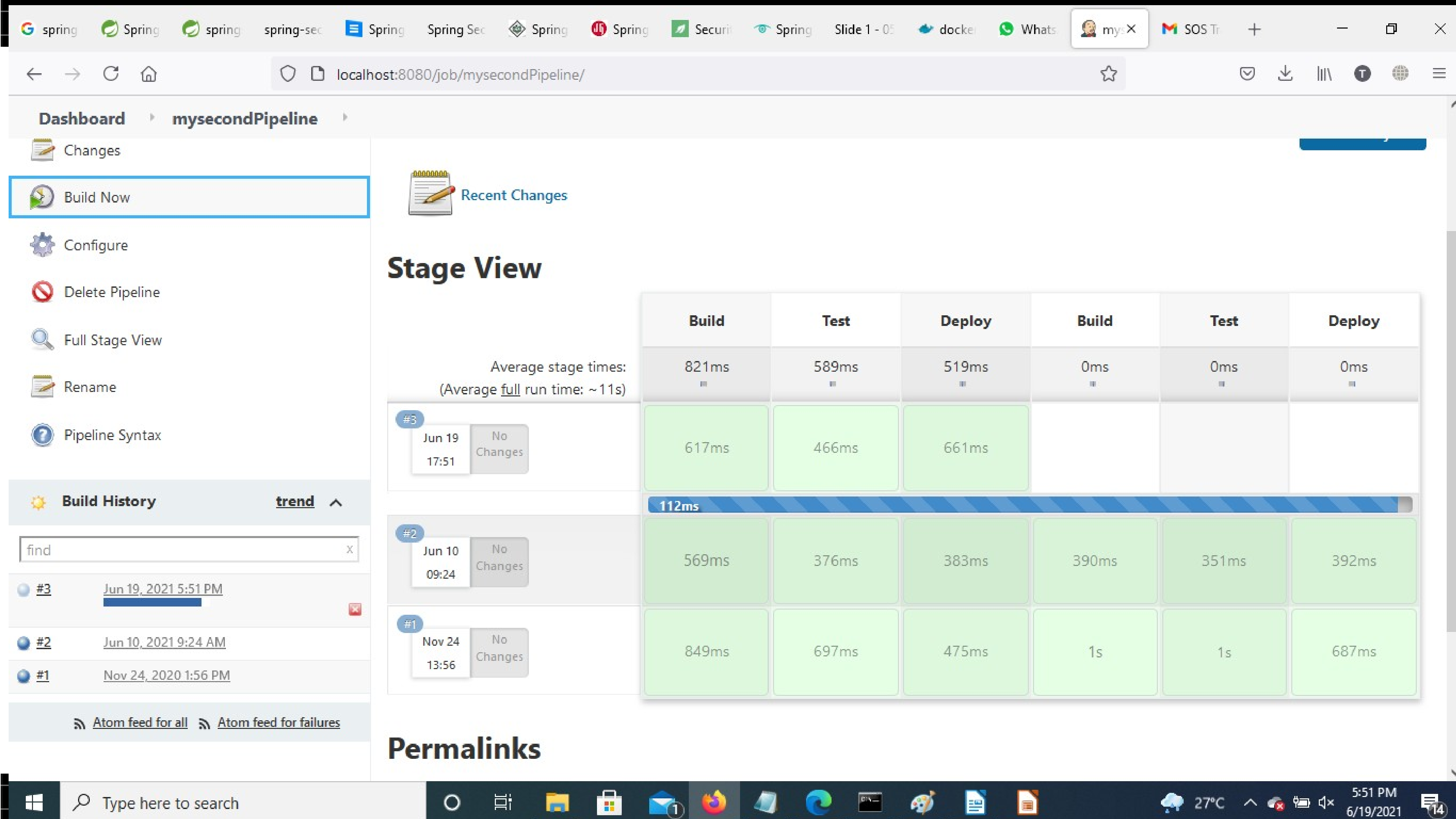

localhost:8080/job/mysecondPipeline/

Pipeline mysecondPipeline

Stage View

	Build	Test	Deploy	Build	Test	Deploy
Average stage times: (Average full run time: ~11s)	872ms	620ms	484ms	0ms	0ms	0ms
#2 Jun 10 09:24 No Changes	569ms	376ms	383ms	390ms	351ms	392ms
#1 Nov 24 13:56 No Changes	849ms	697ms	475ms	1s	1s	687ms

#2 Jun 10, 2021 9:24 AM



localhost:8080/job/mysecondPipeline/

Dashboard › mysecondPipeline ›

Changes
Build Now
Configure
Delete Pipeline
Full Stage View
Rename
Pipeline Syntax

Build History trend

find

#3 Jun 19, 2021 5:51 PM
#2 Jun 10, 2021 9:24 AM
#1 Nov 24, 2020 1:56 PM

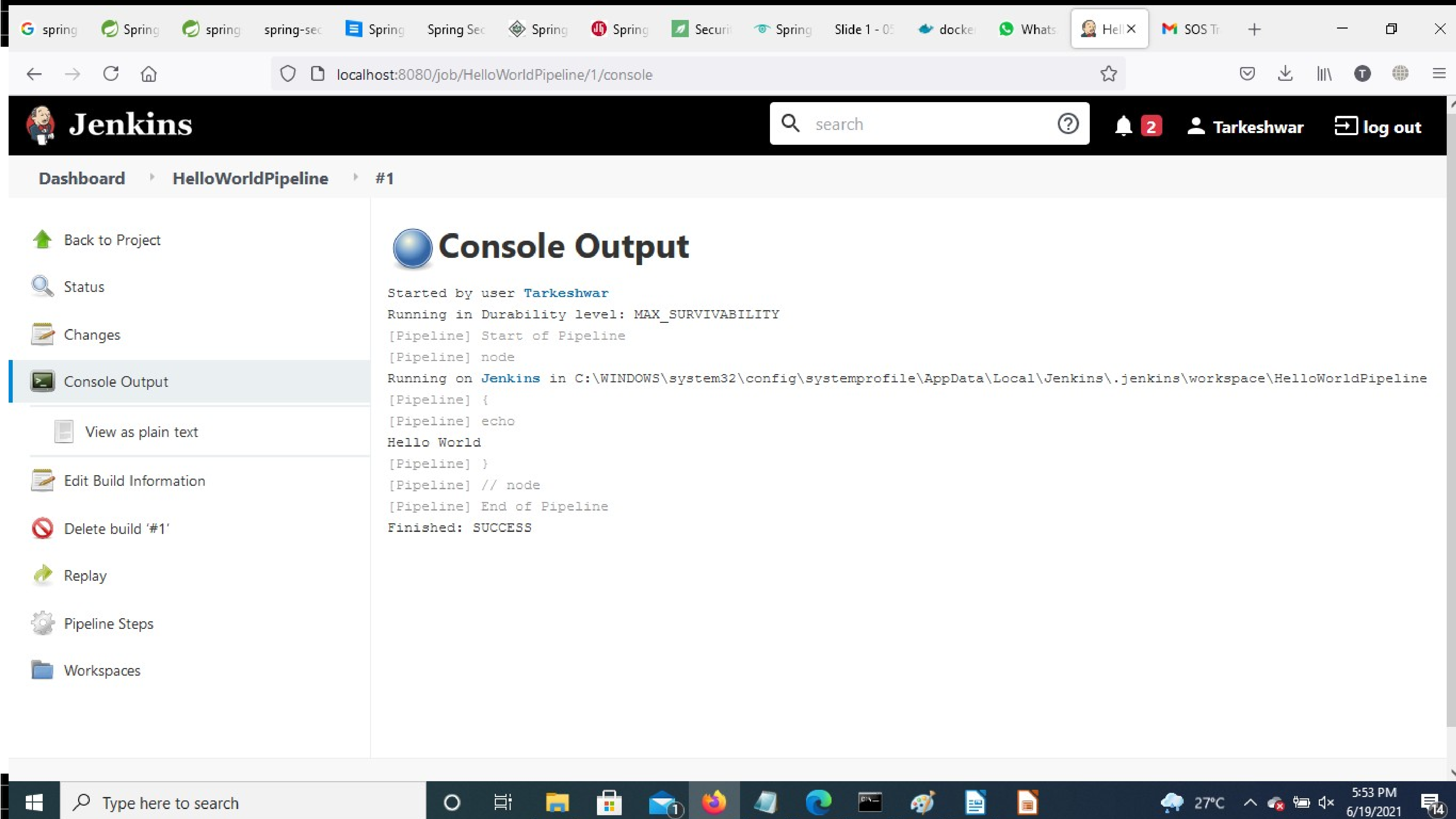
Atom feed for all Atom feed for failures

Recent Changes

Stage View

	Build	Test	Deploy	Build	Test	Deploy
Average stage times: (Average full run time: ~11s)	821ms	589ms	519ms	0ms	0ms	0ms
#3 Jun 19 17:51 No Changes	617ms	466ms	661ms			
	112ms					
#2 Jun 10 09:24 No Changes	569ms	376ms	383ms	390ms	351ms	392ms
#1 Nov 24 13:56 No Changes	849ms	697ms	475ms	1s	1s	687ms

Permalinks



Console Output

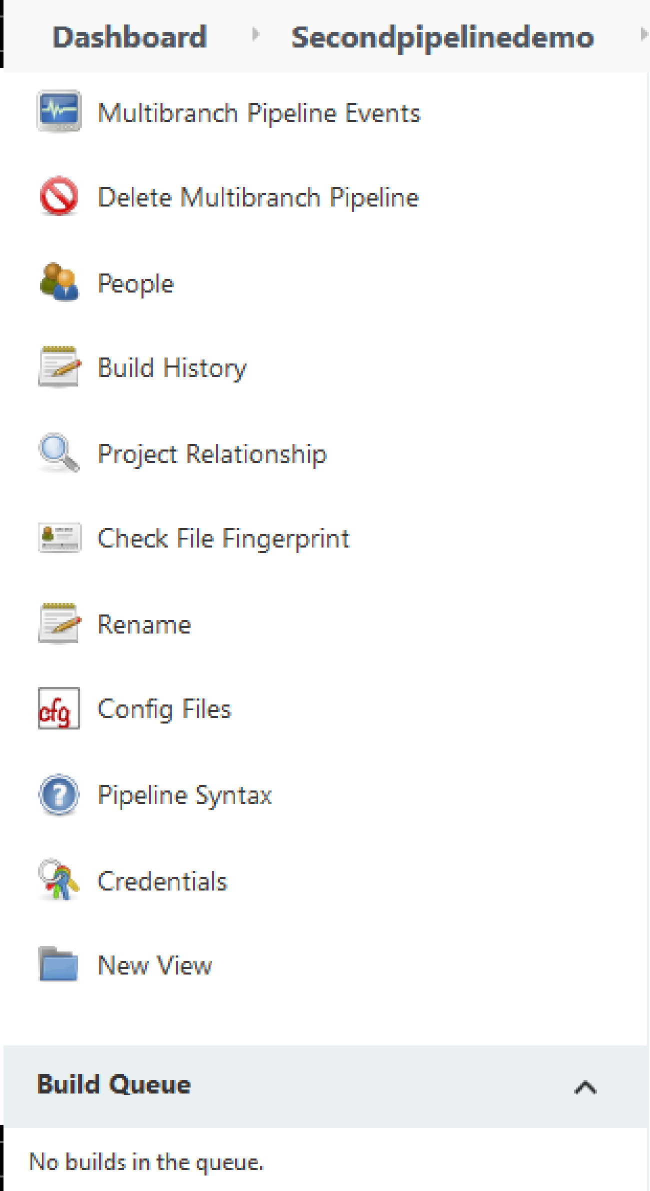
```
Started by user Tarkeshwar
Running in Durability level: MAX_SURVIVABILITY
[Pipeline] Start of Pipeline
[Pipeline] node
Running on Jenkins in C:\WINDOWS\system32\config\systemprofile\AppData\Local\Jenkins\.jenkins\workspace\HelloWorldPipeline
[Pipeline] {
[Pipeline] echo
Hello World
[Pipeline] }
[Pipeline] // node
[Pipeline] End of Pipeline
Finished: SUCCESS
```

Defining a Pipeline in SCM

- Complex Pipelines are hard to write and maintain within the text area of the Pipeline configuration page. To make this easier, Pipeline can also be written in a text editor and checked into source control as a Jenkinsfile which Jenkins can load via the Pipeline Script from SCM option.
- Select Pipeline script from SCM when defining the Pipeline. With the Pipeline script from SCM option selected, you do not enter any Groovy code in the Jenkins UI;
- you just indicate by specifying a path where in source code you want to retrieve the pipeline from. When you update the designated repository, a new build is triggered, as long as the Pipeline is configured with an SCM polling trigger.

Built-in Documentation

- The built-in documentation can be found globally at: **localhost:8080/pipeline-syntax/**, assuming you have a Jenkins instance running on localhost port 8080.
- The same documentation is also linked as Pipeline Syntax in the side-bar for any configured Pipeline project



Snippet Generator

- Navigate to the Pipeline Syntax link (referenced above) from a configured Pipeline, or at **localhost:8080/pipeline-syntax**.
- Select the desired step in the Sample Step dropdown menu
- Use the dynamically populated area below the Sample Step dropdown to configure the selected step.
- Click Generate Pipeline Script to create a snippet of Pipeline which can be copied and pasted into a Pipeline

Global Variable Reference

- **env** Environment variables accessible from Scripted Pipeline, for example: env.PATH or env.BUILD_ID. Consult the built-in Global Variable Reference for a complete, and up to date, list of environment variables available in Pipeline.
- **params** Exposes all parameters defined for the Pipeline as a read-only Map, for example: params.MY_PARAM_NAME.
- **currentBuild** May be used to discover information about the currently executing Pipeline, with properties such as currentBuild.result, currentBuild.displayName, etc. Consult the built-in Global Variable

Using a Jenkinsfile

- Using a text editor, ideally one which supports Groovy syntax highlighting, create a new Jenkinsfile in the root directory of the project.
- The Declarative Pipeline example above contains the minimum necessary structure to implement a continuous delivery pipeline.
- The agent directive, which is required, instructs Jenkins to allocate an executor and workspace for the Pipeline.
- Without an agent directive, not only is the Declarative Pipeline not valid, it would not be capable of doing any work! By default the agent directive ensures that the source repository is checked out and made available for steps in the subsequent stages.

Global Variable Reference

```
// Declarative //
pipeline {
  agent any
  stages {
  stage('Build') {
     steps {
         sh 'make' ①
         archiveArtifacts artifacts: '**/target/*.jar', fingerprint: true ②
      }
    }
  }
}
// Script //
node {
  stage('Build') {
  sh 'make' ①
  archiveArtifacts artifacts: '**/target/*.jar', fingerprint: true ②
  }
}
```

Sample Jenkins Pipeline

```
pipeline {
    agent any
    stages {
       stage('build') {
          steps {
             echo 'Notify GitLab'
             updateGitlabCommitStatus name: 'build', state: 'pending'
             echo 'build step goes here'
             updateGitlabCommitStatus name: 'build', state: 'success'
          }
       }
       stage(test) {
           steps {
               echo 'Notify GitLab'
               updateGitlabCommitStatus name: 'test', state: 'pending'
               echo 'test step goes here'
               updateGitlabCommitStatus name: 'test', state: 'success'

           }
       }
    }
 }
```

Java Jenkins Pipeline

```
pipeline {
    agent any
    stages {
      stage('build') {
        steps {
          echo 'Notify GitLab'
          updateGitlabCommitStatus name: 'build', state: 'pending'
          echo 'build step goes here'
          updateGitlabCommitStatus name: 'build', state: 'success'
        }
      }
      stage(test) {
         steps {
            echo 'Notify GitLab'
            updateGitlabCommitStatus name: 'test', state: 'pending'
            echo 'test step goes here'
            updateGitlabCommitStatus name: 'test', state: 'success'

         }
      }
    }
 }
```

Python Jenkins Pipeline

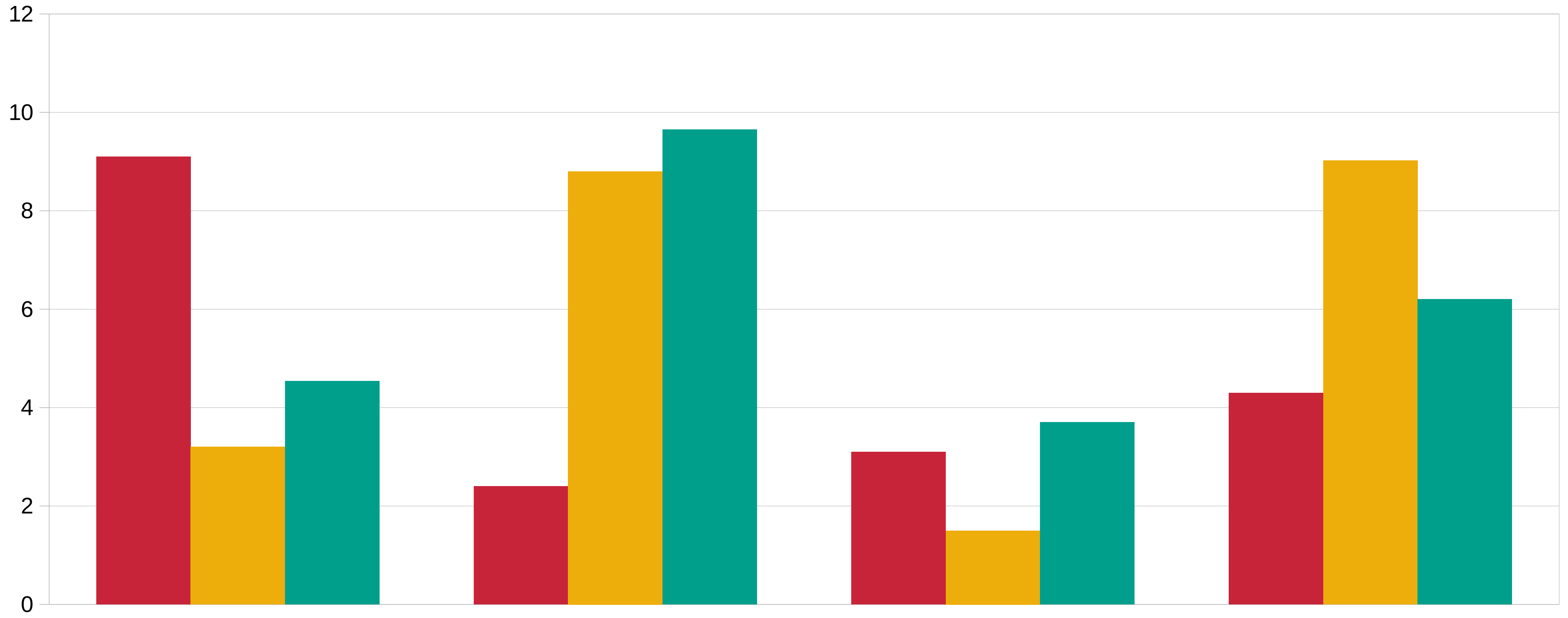
```
pipeline {
   agent any
   stages {
     stage('build') {
       steps {
         echo 'Notify GitLab'
         updateGitlabCommitStatus name: 'build', state: 'pending'
         echo 'build step goes here'
         updateGitlabCommitStatus name: 'build', state: 'success'
       }
     }
     stage(test) {
        steps {
           echo 'Notify GitLab'
           updateGitlabCommitStatus name: 'test', state: 'pending'
           echo 'test step goes here'
           updateGitlabCommitStatus name: 'test', state: 'success'

        }
     }
   }
 }
```

Jenkins CI pipeline integration

- Using a text editor, ideally one which supports Groovy syntax highlighting, create a new Jenkinsfile in the root directory of the project.
- The Declarative Pipeline example above contains the minimum necessary structure to implement a continuous delivery pipeline.
- The agent directive, which is required, instructs Jenkins to allocate an executor and workspace for the Pipeline.
- Without an agent directive, not only is the Declarative Pipeline not valid, it would not be capable of doing any work! By default the agent directive ensures that the source repository is checked out and made available for steps in the subsequent stages.

Global Variable Reference

```
// Declarative //
pipeline {
  agent any
  stages {
  stage('Build') {
     steps {
         sh 'make' ①
         archiveArtifacts artifacts: '**/target/*.jar', fingerprint: true ②
      }
    }
  }
}
// Script //
node {
  stage('Build') {
  sh 'make' ①
  archiveArtifacts artifacts: '**/target/*.jar', fingerprint: true ②
  }
}
```



12
10
8
6
4
2
0