DevOps Practical 1: Version control system and Git	
Name: Hanif Barbhuiya	**Roll number: KCTBCS009**
Date:	**Sign:**

Q. What is the Version Control System?

Version control, also known as source control, is the practice of tracking and managing changes to software code. Version control systems are software tools that help software teams manage changes to source code over time. As development environments have accelerated, version control systems help software teams work faster and smarter. They are especially useful for DevOps teams since they help them to reduce development time and increase successful deployments.

Q. Define Git.

Git is a distributed version control system that enables software development teams to have multiple local copies of the project's codebase independent of each other. These copies, or branches, can be created, merged, and deleted quickly, empowering teams to experiment, with little compute cost, before merging into the main branch. Git is known for its speed, workflow compatibility, and open source foundation.

Most Git actions only add data to the database, and Git makes it easy to undo changes during the three main states.

Git has three file states: modified, staged, and committed.

- A modified file has been changed but isn't committed to the database yet.
- A staged file is set to go into the next commit.
- When a file is committed, the data has been stored in the database.

With Git, software teams can experiment without fearing that they'll create lasting damage to the source code, because teams can always revert to a previous version if there are any problems.

How Git is used in the Version Control System.

Git is a version control system that helps you to manage code and its different versions. It helps to manage different versions of the code base and allows different team members to work on it. In itself git can be used inside a local network like an office but there are cloud providers like github that allow you to manage git repositories on the cloud.

Git can be used on the command line or through the graphical user interface. First an initial empty repository is created where all the code that is meant to be tracked is added once added all the changes made will be tracked.

These files then can be added to remote repositories for storage like on github, bitbucket or gitlab.

Install Git

1. Browse to the official Git website: https://git-scm.com/downloads

2. Click the download link for Windows and allow the download to complete.

3. Browse to the download location (or use the download shortcut in your browser). Double-click the file to extract and launch the installer.

4. Allow the app to make changes to your device by clicking Yes on the User Account Control dialog that opens.

5. Review the GNU General Public Licence, and when you're ready to install, click Next.

6. The installer will ask you for an installation location. Leave the default, unless you have reason to change it, and click Next.

7. A component selection screen will appear. Leave the defaults unless you have a specific need to change them and click Next.

8. The installer will offer to create a start menu folder. Simply click Next.

9. Select a text editor you'd like to use with Git. Use the drop-down menu to select Notepad++ (or whichever text editor you prefer) and click Next.

10. The next step allows you to choose a different name for your initial branch. The default is 'master.' Unless you're working in a team that requires a different name, leave the default option and click Next.

11. This installation step allows you to change the PATH environment. The PATH is the default set of directories included when you run a command from the command line. Leave this on the middle (recommended) selection and click Next.

12. The installer now asks which SSH client you want Git to use. Git already comes with its own SSH client, so if you don't need a specific one, leave the default option and click Next.

13. The next option relates to server certificates. Most users should use the default. If you're working in an Active Directory environment, you may need to switch to Windows Store certificates. Click Next.

14. The next selection converts line endings. It is recommended that you leave the default selection. This relates to the way data is formatted and changing this option may cause problems. Click Next.

15. Choose the terminal emulator you want to use. The default MinTTY is recommended, for its features. Click Next.

16. The installer now asks what the git pull command should do. The default option is recommended unless you specifically need to change its behaviour. Click Next to continue with the installation.

17. Next you should choose which credential helper to use. Git uses credential helpers to fetch or save credentials. Leave the default option as it is the most stable one, and click Next.

18. The default options are recommended, however this step allows you to decide which extra option you would like to enable. If you use symbolic links, which are like shortcuts for the command line, tick the box. Click Next.

19. Depending on the version of Git you're installing, it may offer to install experimental features. At the time this article was written, the options to include support for pseudo controls and a built-in file system monitor were offered. Unless you are feeling adventurous, leave them unchecked and click Install.

20. Once the installation is complete, tick the boxes to view the Release Notes or Launch Git Bash, then click Finish.

DevOps Practical 2: Git commands and Git flow with diagrams	
Name: Hanif Barbhuiya	**Roll number: KCTBCS009**
Date:	**Sign:**

List of git commands

Git init	This command is used to start a new repository.
Git pull	This command is used to pull changes from a remote repository
Git push	This command is used to push changes to remote repository
Git stash	This command temporarily stores all the modified tracked files.
Git checkout	This command is used to move head to a new branch
Git branch	You can create a new branch with the help of the git branch command.
Git remote	The command lets you create, view, and delete connections to other repositories
Git rebase	It is used to apply commits from distinct branches into a final commit
Git merge	You can merge branches with the help of the git merge command.
Git clone	This command is used to obtain a repository from an existing URL.
Git commit	This command records or snapshots the file permanently in the version history.
Git add	This command is used to add files to repositories to track changes of
Git log	This command gives a list of all changes made in a repository

Git status	Shows the status of tracked files: newly added, modified or deleted
Git reset	This command unstages the file, but it preserves the file contents.
Git config	This command sets the author name and email address respectively to be used with your commits.
Git fetch	git fetch is a primary command used to download contents from a remote repository.
Git revert	The git revert command is used to apply revert operation. It is an undo type command.
Git rm	This command deletes the file from your working directory and stages the deletion.
Git show	This command shows the metadata and content changes of the specified commit.

Git Flow

Gitflow is a legacy Git workflow that was originally a disruptive and novel strategy for managing Git branches. Gitflow has fallen in popularity in favour of trunk-based workflows, which are now considered best practices for modern continuous software development and DevOps practices. Gitflow also can be challenging to use with CI/CD.

What is Gitflow?

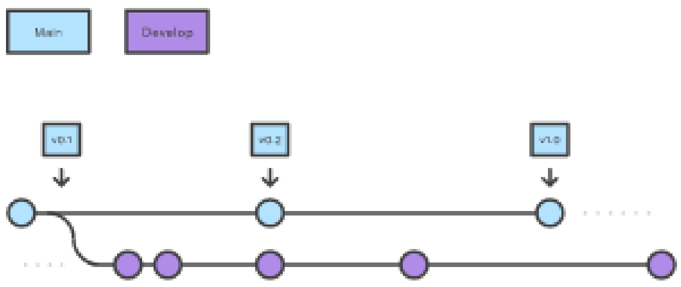
Giflow is an alternative Git branching model that involves the use of feature branches and multiple primary branches. It was first published and made popular by Vincent Driessen at nvie. Compared to trunk-based development, Giflow has numerous, longer-lived branches and larger commits. Under this model, developers create a feature branch and delay merging it to the main trunk branch until the feature is complete. These long-lived feature branches require more collaboration to merge and have a higher risk of deviating from the trunk branch. They can also introduce conflicting updates.

Gitflow can be used for projects that have a scheduled release cycle and for the DevOps best practice of continuous delivery. This workflow doesn’t add any new concepts or commands beyond what’s required for the Feature Branch Workflow. Instead, it assigns very specific roles to different branches and defines how and when they should interact. In addition to feature branches, it uses individual branches for preparing, maintaining, and recording releases. Of course, you also get to leverage all the benefits of the Feature Branch Workflow: pull requests, isolated experiments, and more efficient collaboration.

How it works

Develop and main branches

Instead of a single main branch, this workflow uses two branches to record the history of the project. The main branch stores the official release history, and the develop branch serves as an integration branch for features. It's also convenient to tag all commits in the main branch with a version number.



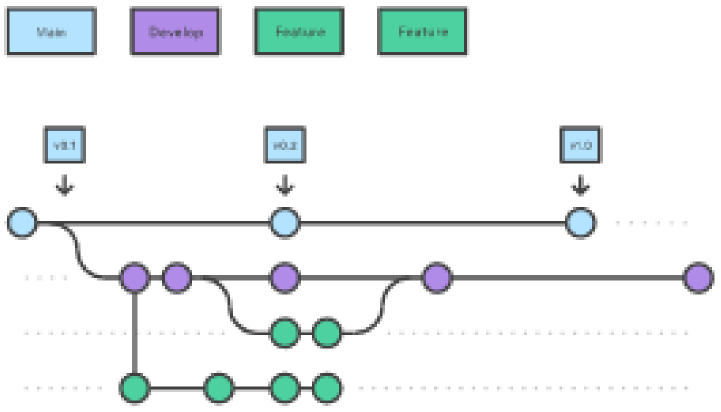
The first step is to complement the default main with a develop branch. A simple way to do this is for one developer to create an empty develop branch locally and push it to the server:

git branch develop

git push -u origin develop

This branch will contain the complete history of the project, whereas main will contain an abridged version. Other developers should now clone the central repository and create a tracking branch for development.

Feature branches



Each new feature should reside in its own branch, which can be pushed to the central repository for backup/collaboration. But, instead of branching off of main, feature branches use develop as their parent branch. When a feature is complete, it gets merged back into develop. Features should never interact directly with main.

Note that feature branches combined with the develop branch is, for all intents and purposes, the Feature Branch Workflow. But, the Gitflow workflow doesn't stop there.

Feature branches are generally created off to the latest develop branch.

Creating a feature branch

Without the git-flow extensions:

git checkout develop

git checkout -b feature_branch

When using the git-flow extension:

git flow feature start feature_branch

Continue your work and use Git like you normally would.

Finishing a feature branch

When you’re done with the development work on the feature, the next step is to merge the feature_branch into develop.

Without the git-flow extensions:

```
git checkout develop
```

```
git merge feature_branch
```

Using the git-flow extensions:

```
git flow feature finish feature_branch
```

DevOps Practical 3: Staging and commit commands	
Name: Hanif Barbhuiya	**Roll number: KCTBCS009**
Date:	**Sign:**

Q3. Explain and Implement all the staging and commit commands in Git.

Git Init

the git init command is used to create new version-controlled projects.

```
PS C:\Users\ADMIN\Desktop> git init
Initialized empty Git repository in C:/Users/ADMIN/Desktop/.git/
```

Git Add

This command is used to add files to a repository.

```
PS C:\Users\ADMIN\Desktop> git add .
```

Git Commit

This command is used to commit changes done to files being tracked in a repository

There are two ways to use this command.

> git commit -m: This command changes the head. It records or snapshots the file permanently in the version history with a message.

```
PS C:\Users\ADMIN\Desktop> git commit -m "First"
[master (root-commit) 9a760f8] First
 1 files changed, 1 insertion(+)
 create mode 160000 Frontend-Projects
```

> git commit -a: This command commits any files added in the repository with git add and also commits any files you've changed since then.

```
PS C:\Users\ADMIN\Desktop> git commit -a
```

Git Clone

This command is used to make a copy of a repository from an existing URL. If I want a local copy of my repository from GitHub, this command allows creating a local copy of that repository on your local directory from the repository URL.

```
PS C:\Users\ADMIN\Desktop> git clone https://github.com/1Hanif1/Frontend-Projects.git

Cloning into 'Frontend-Projects'...

remote: Enumerating objects: 1124, done.

remote: Counting objects: 100% (1124/1124), done.

remote: Compressing objects: 100% (693/693), done.

remote: Total 1124 (delta 500), reused 1022 (delta 403), pack-reused 0

Receiving objects:  97% (1091/1124), 11.08 MiB | 2.20 MiB/s

Receiving objects: 100% (1124/1124), 11.79 MiB | 2.21 MiB/s, done.

Resolving deltas: 100% (500/500), done.
```

Git Stash

git stash temporarily shelves (or stashes) changes you've made to your working copy so you can work on something else, and then come back and re-apply them later on.

```
PS C:\Users\ADMIN\Desktop> git status

On branch master

Changes to be committed:

  (use "git restore --staged <file>..." to unstage)

        new file:   edit.txt

PS C:\Users\ADMIN\Desktop> git stash

Saved working directory and index state WIP on master: 9a760f8 First

PS C:\Users\ADMIN\Desktop> git status

On branch master

nothing to commit, working tree clean
```

Git Ignore

The git ignore is a special hidden file in a git repo in which all files that you do not wish the git system to track changes of are specified. There is no explicit git ignore command in git bash

Git Fork

A fork in Git is simply a copy of an existing repository in which the new owner disconnects the codebase from previous committers. A fork often occurs when a developer becomes dissatisfied or disillusioned with the direction of a project and wants to detach their work from that of the original project. When a git fork occurs, previous contributors will not be able to commit code to the new repository without the owner giving them access to the forked repo, either by providing developers the publicly accessible Git URL, or by providing explicit access through user permission in tools like GitHub or GitLab. There is no explicit git fork command. If you have a Git repository on your personal computer, you can create a fork simply by copying the Git repo to a new folder and then removing any remote references in the Git config file. That will create a isolated and independent Git fork that will no longer synchronize with the original codebase. It's really that simple.

Git Repository

In Git, the repository is like a data structure used by VCS to store metadata for a set of files and directories. It contains the collection of the files as well as the history of changes made to those files. Repository in Git is considered as your project folder. A repository has all the project-related data. Distinct projects have distinct repositories.

There are two ways to obtain a repository. They are as follows:

- Create a local repository and make it as Git repository.
- Clone a remote repository (already exists on a server).

To create a git repository you need to execute the following command:

```
PS C:\Users\ADMIN\Desktop> git init

Initialized empty Git repository in C:/Users/ADMIN/Desktop/.git/
```

Git Index

The Git index is a critical data structure in Git. It serves as the "staging area" between the files you have on your filesystem and your commit history. When you run git add, the files from your working directory are hashed and stored as objects in the index, leading them to be "staged changes". When you run git commit, the staged changes as stored in the index are used to create that new commit. When you run git checkout, Git takes the data from a commit and writes it to the working directory and the index.

Git Head

When working with Git, only one branch can be checked out at a time - and this is what's called the "HEAD" branch. Often, this is also referred to as the "active" or "current" branch.

Git makes note of this current branch in a file located inside the Git repository, in .git/HEAD.

In rare cases, the HEAD file does NOT contain a branch reference, but a SHA-1 value of a specific revision. This happens when you checkout a specific commit, tag, or remote branch. Your repository is then in a state called Detached HEAD.

Git Origin Master

When we want to contribute to a git project, we need to make sure how to manage the remote repositories. One can push and pull data from a remote repository when you need to share work with teams. Origin and Master are two different terminologies used when working and managing the git projects.

- Origin is the name used for the remote repository.
- Master is the name of the branch.

The term "git origin master" is used in the context of a remote repository. It is used to deal with the remote repository. The term origin comes from where repository original situated and master stands for the main branch. Let's understand both of these terms in detail.

Master is a naming convention for Git branch. It's a default branch of Git. After cloning a project from a remote server, the resulting local repository contains only a single local branch. This branch is called a "master" branch. It means that "master" is a repository's "default" branch.

Git Remote

The git remote command lets you create, view, and delete connections to other repositories. Remote connections are more like bookmarks rather than direct links into other repositories. Instead of providing real-time access to another repository, they serve as convenient names that can be used to reference a not-so-convenient URL.

PS C:\Users\ADMIN\Desktop> git remote

List the remote connections you have to other repositories.

PS C:\Users\ADMIN\Desktop> git remote -v

Same as the above command, but include the URL of each connection.

PS C:\Users\ADMIN\Desktop> git remote add <name> <url>

Create a new connection to a remote repository. After adding a remote, you’ll be able to use ＜name＞ as a convenient shortcut for ＜url＞ in other Git commands.

PS C:\Users\ADMIN\Desktop> git remote rm <name>

DevOps Practical 4: git commands for undoing changes	
Name: Hanif Barbhuiya	**Roll number: KCTBCS009**
Date:	**Sign:**

Git Checkout

In Git, the term checkout is used for the act of switching between different versions of a target entity. The git checkout command is used to switch between branches in a repository. Be careful with your staged files and commits when switching between branches.

The git checkout command operates upon three different entities which are files, commits, and branches. Sometimes this command can be dangerous because there is no undo option available on this command.

It checks the branches and updates the files in the working directory to match the version already available in that branch, and it forwards the updates to Git to save all new commits in that branch.

<u>List all branches</u>

```
PS C:\Users\ADMIN\Internship\ob-fe-customer> git branch
  dev
* hanif
```

<u>Change branch</u>

```
PS C:\Users\ADMIN\Internship\ob-fe-customer> git checkout dev
Switched to branch 'dev'
Your branch is behind 'origin/dev' by 116 commits, and can be fast-forwarded.
  (use "git pull" to update your local branch)
```

<u>Create new branch</u>

```
PS C:\Users\ADMIN\Internship\ob-fe-customer> git checkout -b temp
Switched to a new branch 'temp'
```

Git Revert

In Git, the term revert is used to revert some changes. The git revert command is used to apply revert operation. It is an undo type command. However, it is not a traditional undo alternative. It does not delete any data in this process; instead, it will create a new change with the opposite effect and thereby undo the specific commit. Generally, git revert is a commit. It can be useful for tracking bugs in the project. If you want to remove something from history then git revert is a wrong choice.

<u>Get list of all commits made</u>

```
PS C:\Users\ADMIN\Internship\ob-fe-customer> git log
commit 44371715ed1855d4491586526df565ee65d01279 (HEAD -> temp, origin/dev, hanif)
Merge: d1116fb 8013b86
Author: OyeBusy Next <oyebusy.next@gmail.com>
Date:   Fri Sep 23 15:52:29 2022 +0000
    Merge branch 'hanif' into 'dev'
    Profile: Address section and Logout function
```

See merge request on-next/frontend-web-applications/ob-fe-customer!9

Revert to previous commit

```
PS C:\Users\ADMIN\Internship\ob-fe-customer> git revert
44371715ed1855d4491586526df565ee65d01279
```

Git Reset

The term reset stands for undoing changes. The git reset command is used to reset the changes. If we say in terms of Git, then Git is a tool that resets the current state of HEAD to a specified state. It is a sophisticated and versatile tool for undoing changes. It acts as a time machine for Git. You can jump up and forth between the various commits. Each of these reset variations affects specific trees that git uses to handle your file in its content.

Hard Reset

```
PS C:\Users\ADMIN\Internship\ob-fe-customer> git reset –hard

HEAD is now at 44371715ed Revert 'CSS File'
```

This option will reset the changes and match the position of the Head before the last changes. It will remove the available changes from the staging area

Mixed Reset

A mixed option is a default option of the git reset command. If we would not pass any argument, then the git reset command is considered as --mixed as default option. A mixed option updates the ref pointers. The staging area also reset to the state of a specific commit. The undone changes transferred to the working directory.

```
PS C:\Users\ADMIN\Internship\ob-fe-customer> git reset --mixed

PS C:\Users\ADMIN\Internship\ob-fe-customer>git status

on branch test2

untracked files :

(use "git add <file>... " to include in what will1 be committed)

newfile2. txt

nothing added to commit but untracked files present (use "git add" to t
```

Soft reset

The soft option does not touch the index file or working tree at all, but it resets the Head as all options do. When the soft mode runs, the refs pointers are updated, and the resets stop there. It will act as a git amend command. It is not an authoritative command. Sometimes developers considered it as a waste of time.

```
PS C:\Users\ADMIN\Internship\ob-fe-customer> git reset --soft 2c5a8820091654
```

Git Rm

In Git, the term rm stands for remove. It is used to remove individual files or a collection of files. The key function of git rm is to remove tracked files from the Git index. Additionally, it can be used to remove files from both the working directory and staging index. The files being removed must be ideal for the branch to remove. No updates to their contents can be staged in the index. Otherwise, the removal process can be complex, and sometimes it will not happen. But it can be done forcefully by -f option.

<u>Removing a file</u>

```
PS C:\Users\ADMIN\Internship\ob-fe-customer> git rm newfile. txt

rm 'newfile. txt"

PS C:\Users\ADMIN\Internship\ob-fe-customer> git status

on branch master

Changes to be committed:

(use "git restore --staged <file>..." to unstage)

deleted:

newfile. txt
```

<u>Removing cached file</u>

```
PS C:\Users\ADMIN\Internship\ob-fe-customer>git rm--cached newfile1. txt

rm newfilel.txt'

PS C:\Users\ADMIN\Internship\ob-fe-customer> git status

on branch master

Changes to be committed:

(use "git restore --staged <file>... " to unstage)

deleted:

newfile1. txt

untracked files:

(use "git add <file>... " to include in what wi11 be committed)

newfile1. txt
```

DevOps Practical 5: git commands for inspecting changes	
Name: Hanif Barbhuiya	**Roll number: KCTBCS009**

Date:	**Sign:**

Git Log

Git log is a utility tool to review and read a history of everything that happens to a repository. Multiple options can be used with a git log to make history more specific.

Generally, the git log is a record of commits. A git log contains the following data:

- A commit hash, which is a 40 character checksum data generated by SHA (Secure Hash Algorithm) algorithm. It is a unique number.
- Commit Author metadata: The information of authors such as author name and email.
- Commit Date metadata: It's a date timestamp for the time of the commit.
- Commit title/message: It is the overview of the commit given in the commit message

```
PS C:\Users\ADMIN\Internship\ob-fe-customer> git log

commit 44371715ed1855d4491586526df565ee65d01279 (HEAD -> temp, origin/dev, hanif)

Merge: d1116fb 8013b86

Author: OyeBusy Next <oyebusy.next@gmail.com>

Date:   Fri Sep 23 15:52:29 2022 +0000

    Merge branch 'hanif' into 'dev'

    Profile: Address section and Logout function

    See merge request on-next/frontend-web-applications/ob-fe-customer!9
```

Git Log Oneline

The one line option is used to display the output as one commit per line. It also shows the output in brief like the first seven characters of the commit SHA and the commit message.

```
PS C:\Users\ADMIN\Internship\ob-fe-customer> git log --oneline

4437171 (HEAD -> temp, origin/dev, hanif) Merge branch 'hanif' into 'dev'

8013b86 (origin/hanif) Css changes

607bab5 Minor Css changes

056896b Added Dustbin icon and minor changes to profile store

06d77d6 Routing to main page if user not authenticated

e83 maf Delete Address feature

2576717 Changed CSS and added props

ee24c1e Added logout feature
```

Git Log Stat

The log command displays the files that have been modified. It also shows the number of lines and a summary line of the total records that have been updated.

Generally, we can say that the stat option is used to display

- the modified files,
- The number of lines that have been added or removed
- A summary line of the total number of records changed
- The lines that have been added or removed.

```
PS C:\Users\ADMIN\Internship\ob-fe-customer> git log --stat

commit 44371715ed1855d4491586526df565ee65d01279 (HEAD -> temp, origin/dev, hanif)

Merge: d1116fb 8013b86

Author: OyeBusy Next <oyebusy.next@gmail.com>

Date:   Fri Sep 23 15:52:29 2022 +0000

    Merge branch 'hanif' into 'dev'

    Profile: Address section and Logout function

    See merge request on-next/frontend-web-applications/ob-fe-customer!9
```

Git Diff

Git diff is a command-line utility. It's a multi-use Git command. When it is executed, it runs a diff function on Git data sources. These data sources can be files, branches, commits, and more. It is used to show changes between commits, commit, and working tree, etc. ș

It compares the different versions of data sources. The version control system stands for working with a modified version of files. So, the diff command is a useful tool for working with Git.

However, we can also track the changes with the help of git log command with option -p. The git log command will also work as a git diff command.

<u>Track the changes that have not been staged.</u>

```
PS C:\Users\ADMIN\Internship\ob-fe-customer> git diff

diff --git a/newfile1. txt b/newfi le1. txt

index ade63b7. .41a6a9c 100644

--- a/new file. txt

+++ b/newfilel.txt

@@-3,3 +3,4 @a i am on test2 branch.

91t committee

git commit2

git merge demo

changes are made to understand the git diff comm
```

<u>Track the changes that have staged but not committed</u>

```
PS C:\Users\ADMIN\Internship\ob-fe-customer> git diff --staged
diff--git a/new file 1. txt b/newfile1. txt
index ade63b7 . . 41a6a9c 100644
-- a/newti lel.txt
+++ b/newfilel.txt
@-3,3 +3,4 @a i am on test2 branch.
git commit
git commit2
git merge demo
-changes are made to understand the git diff command.
```

<u>Track the changes after committing a file</u>

```
PS C:\Users\ADMIN\Internship\ob-fe-customer> git diff HEAD
diff-git a/newfile1. txt b/newfile1. txt
index 41a6a9c. .e14624d 100644
--- a/newfilel.txt
+++ b/newti lel. txt
-4,3 +,4 G git commit
git commit2
git merge demo
changes are made to understand the git diff command.
+changes are made after committing the file.
```

<u>Track the changes between two commits</u>

```
PS C:\Users\ADMIN\Internship\ob-fe-customer> git diff e553fc08cb f1ddc7c9e7
diff --git a/.gitignore b/.gitignore
deleted file mode 100644
index 8527c9c. . 0000000
-- a/.gitignore
ttt /dev/null]
```

```
,2 +0,0 @a
.txt
/newfo l der/
newline at end of file
diff --git a/newfilel. txt b/newfile1. txt
index 41a6a9c. . ade63b7 100644
*-- a/newti lel.txt
+ b/newfilel.txt
a-3,4 +3,3 a i am on test2 branch.
git commit
git commit 2
```

Git Status

The git status command is used to display the state of the repository and staging area. It allows us to see the tracked, untracked files and changes. This command will not show any commit records or information. Mostly, it is used to display the state between Git Add and Git commit commands. We can check whether the changes and files are tracked or not.

```
PS C:\Users\ADMIN\Internship\ob-fe-customer> touch demo file
PS C:\Users\ADMIN\Internship\ob-fe-customer> git status
on branch master
untracked files :
(use "git add <file..." to include in what wi11 be committed)
demo file
nothing added to commit but untracked files present (use "git add" to
track)
```

DevOps Practical 6: git commands for branching and merging	
Name: Hanif Barbhuiya	**Roll number: KCTBCS009**
Date:	**Sign:**

Git Branch

A branch is a version of the repository that diverges from the main working project. It is a feature available in most modern version control systems. A Git project can have more than one branch. These branches are a pointer to a snapshot of your changes. When you want to add a new feature or fix a bug, you spawn a new branch to summarise your changes. So, it is complex to merge the unstable code with the main code base and also facilitates you to clean up your future history before merging with the main branch.

Git Master Branch

The master branch is a default branch in Git. It is instantiated when the first commit is made on the project. When you make the first commit, you're given a master branch to the starting commit point. When you start making a commit, then the master branch pointer automatically moves forward. A repository can have only one master branch.

Master branch is the branch in which all the changes eventually get merged back. It can be called an official working version of your project.

Create Branch

You can create a new branch with the help of the git branch command. This command will be used as:

```
PS C:\Users\ADMIN\Internship\ob-fe-customer>git branch B1
```

List all branches

```
PS C:\Users\ADMIN\Internship\ob-fe-customer> git branch

B1

* master

PS C:\Users\ADMIN\Internship\ob-fe-customer> git branch --list

B1

master
```

Delete a branch

```
PS C:\Users\ADMIN\Internship\ob-fe-customer> git branch -d B1

Deleted branch B1 (was 554a122) .
```

Switch to another branch

```
PS C:\Users\ADMIN\Internship\ob-fe-customer> git checkout branch4

Switched to branch "branch4"
```

Merge & Merge Conflict

In Git, the merging is a procedure to connect the forked history. It joins two or more development histories together. The git merge command facilitates you to take the data created by the git branch and integrate them into a single branch. Git merge will associate a series of commits into one unified history. Generally, git merge is used to combine two branches. It is used to maintain distinct lines of development; at some stage, you want to merge the changes in one branch. It is essential to understand how merging works in Git.

<u>The git merge command is used to merge the branches.</u>

```
PS C:\Users\ADMIN\Internship\ob-fe-customer> git checkout master

Switched to branch 'master"

Your branch is up to date with 'origin/master.

PS C:\Users\ADMIN\Internship\ob-fe-customer> git merge 2852e020909dfe705 707695 fd6d715
cd723f9540

Updating 4a6693a. . 2852e02

Fast-forward

newfile. txt

newfilel. txt | 1 +

2 files changed, 2 insertions (+)

create mode 100644 newfile. txt

create mode 100644 newfile. txt
```

<u>Merge conflicts</u>

When two branches are trying to merge, and both are edited at the same time and in the same file, Git won't be able to identify which version is to take for changes. Such a situation is called merge conflict. If such a situation occurs, it stops just before the merge commit so that you can resolve the conflicts manually.

```
PS C:\Users\ADMIN\Internship\ob-fe-customer> git add index. htm

PS C:\Users\ADMIN\Internship\ob-fe-customer> git commit -m "edited by user2"

[master 3ee71e0] edited by user2

1 file changed, 1 insertion(+)

PS C:\Users\ADMIN\Internship\ob-fe-customer> git push origin master

To https://github. com/ImDwi vedi1/Git-Example

Rejected)

error : failed to push some refs to 'https ://github. com/ ImDwi ved11/Git-Example'
```

```
hint: Updates were rejected because the remote contains work that you do
hint: not locally. This is usually caused by another repository pushing
hint: to the same ref. You may want to first integrate the remote changes
hint: (e. g. , 'git pull... ") before pushing again.
hint: See the "Note about fast-forwards" in "git push -help" for detail.
master -> master (fetch first)
```

Resolve Conflict:

To resolve the conflict, it is necessary to know whether the conflict occurs and why it occurs. Git merge tool command is used to resolve the conflict.

Git Rebase

Rebasing is a process to reapply commits on top of another base trip. It is used to apply a sequence of commits from distinct branches into a final commit. It is an alternative to the git merge command. It is a linear process of merging. In Git, the term rebase is referred to as the process of moving or combining a sequence of commits to a new base commit. Rebasing is very beneficial and it visualises the process in the environment of a feature branching workflow. Generally, it is an alternative to the git merge command. Merge is always a forward changing record. Comparatively, rebase is a compelling history rewriting tool in git. It merges the different commits one by one.

Rebase branch

If we have many commits from distinct branches and want to merge it in one. To do so, we have two choices: either we can merge it or rebase it. It is good to rebase your branch.

```
PS C:\Users\ADMIN\Internship\ob-fe-customer> git rebase master
First, rewinding your head to replay your work on top of it...
Fast-forwarded test to master.
```

Git interactive rebase

Git facilitates with Interactive Rebase; it is a potent tool that allows various operations like edit, rewrite, reorder, and more on existing commits. Interactive Rebase can only be operated on the currently checked out branch. Therefore, set your local HEAD branch at the sidebar. Git interactive rebase can be invoked with rebase command, just type -i along with rebase command. Here 'i' stands for interactive.

```
PS C:\Users\ADMIN\Internship\ob-fe-customer> git rebase -i
hint: Waiting for your editor to close the file..
```

When we perform the git interactive rebase command, it will open your default text editor

The options it contains are: Pick, Reword, Edit, Squash, Fixup, Exec, Break, Drop, Label, Reset, Merge

Pick (-p):

Pick stands here that the commit is included. Order of the commits depends upon the order of the pick commands during rebase. If you do not want to add a commit, you have to delete the entire line.

Reword (-r):

The reword is quite similar to the pick command. The reword option paused the rebase process and provided a chance to alter the commit message. It does not affect any changes made by the commit.

Edit (-e):

The edit option allows for amending the commit. The amending means, commits can be added or changed entirely. We can also make additional commits before rebase continue command. It allows us to split a large commit into the smaller commit; moreover, we can remove erroneous changes made in a commit.

Squash (-s):

The squash option allows you to combine two or more commits into a single commit. It also allows us to write a new commit message for describing the changes.

Fixup (-f):

It is quite similar to the squash command. It discarded the message of the commit to be merged. The older commit message is used to describe both changes.

Exec (-x):

The exec option allows you to run arbitrary shell commands against a commit.

Break (-b):

The break option stops the rebasing at just position. It will continue rebasing later with 'git rebase --continue' command.

Drop (-d):

The drop option is used to remove the commit.

Label (-l):

The label option is used to mark the current head position with a name.

Reset (-t):

The reset option is used to reset head to a label.

DevOps Practical 7: Collaborating commands in Git.	
Name: Hanif Barbhuiya	**Roll number: KCTBCS009**
Date:	**Sign:**

Git Fetch

Git "fetch" Downloads commits, objects and refs from another repository. It fetches branches and tags from one or more repositories. It holds repositories along with the objects that are necessary to complete their histories to keep updated remote-tracking branches.

The "git fetch" command is used to pull the updates from remote-tracking branches. Additionally, we can get the updates that have been pushed to our remote branches to our local machines. As we know, a branch is a variation of our repository's main code, so the remote-tracking branches are branches that have been set up to pull and push from remote repository.

<u>To fetch the remote repository</u>

```
PS C:\Users\ADMIN\Internship\ob-fe-customer> git fetch https://github. com/ImDwi vedi 1/Git-Example.
git

warning : no common commits

remote: Enumerating objects: 6, done.

remote: Counting objects: 100% (6/6), done.

emote: Compressing objects: 100% (4/4), done.

emote : Total 6 (delta 0), reused 0 (delta 0), pack-reused 0

Unpacking objects: 100% (6/6), done.

From https://github. com/ImDwi vedi1/Git-Exampl e

branch

HEAD

-> FETCH_HEAD
```

To fetch a specific branch

```
PS C:\Users\ADMIN\Internship\ob-fe-customer> git fetch https://github. com/ImDwi vedi 1//Git-Example.
git Test
warning : no common commits
remote: Enumerating objects: 9, done.
remote: counting objectS: 100% (9/9), done.
remote: Compr essing_objects: 100% (6/6), done.
remote: Total 9 (delta 1), reused 0 (delta 0), pack-reused 0
unpacking objects: 100% (9/9), done.
From https://github. com/ImDwi video/Git-Example
* branch
Test
-> FETCH_HEAD
```

To fetch all the branches simultaneously

```
PS C:\Users\ADMIN\Internship\ob-fe-customer> git fetch --all
Fetching origin
From https://github.com/ImDwi vedi 1/Git-Example
* [new branch]
[new branch]
master
-> origin/master
->origin/Test
Test
```

To synchronise the local repository

```
PS C:\Users\ADMIN\Internship\ob-fe-customer> git fetch origin
HiMaNshuGHiMaNs hu-pC MINGW64 -/Desktop/Git-Example (master)
Git fetch origin
remote:Enumerating objects: 4, done.
remote: Counting objects: 100% (4/4), done.
remote: Compr essing_objects: 100% (2/2), done.
remote: Total 3 (delta 1), reused 0 (delta 0), pack-reus ed o
```

```
unpacking objects: 100% (3/3), done.

From https://github. com/ImDwi video/Git-Example

*[new branch]

test2

->origin/test2
```

Git Pull

The term pull is used to receive data from GitHub. It fetches and merges changes from the remote server to your working directory. The git pull command is used to pull a repository.

Pull request is a process for a developer to notify team members that they have completed a feature. Once their feature branch is ready, the developer files a pull request via their remote server account. Pull request announces all the team members that they need to review the code and merge it into the master branch.

Pull command

The pull command is used to access the changes (commits)from a remote repository to the local repository. It updates the local branches with the remote-tracking branches. Remote tracking branches are branches that have been set up to push and pull from the remote repository. Generally, it is a collection of the fetch and merge commands. First, it fetches the changes from remote and combines them with the local repository.

Default git pull

```
PS C:\Users\ADMIN\Internship\ob-fe-customer> git pull

emote: Enumerating objects: 4, done.

remote : Counting objects : 100% (4/4) . done.

emote: Compr essing_objects: 100% (3/3), done.

emote: Total 3 (delta 1) , reused o (delta 0), pack-reus ed 0

unpacking objects: 100% (3/3), done.

From https ://github. com/TmDwi vedi 1/GitExample2

find dc7c. .0ala475 master

-> origin/master

updating f1ddc7c. . Oala475

Fast-forward

design2. css | 6 ++++++
1 file changed, 6 insertions (+)

create mode 100644 design2. css
```

Git Pull Remote Branch

Git allows fetching a particular branch. Fetching a remote branch is a similar process, as mentioned above, in git pull command. The only difference is we have to copy the URL of the particular branch we want to pull. To do so, we will select a specific branch.

```
PS C:\Users\ADMIN\Internship\ob-fe-customer>  git pull https://github. com/TmDwi vedi1/GitExampl
e2.git

remote: Enumerating objects : 38, done.

remote: Counting objects: 100% (38/38), done.

remote: Compressing objects: 100% (25/25), _done.

remote: Total 38 (delta 13). reused 19 (delta 7), pack-reused 0

unpacking objects: 100% (38/38), done.

From https://github. com/ImDwi vedi 1/GitExample2

*branch

HEAD

->FETCH_HEAD
```

Git Force Pull

Git force pull allows for pulling your repository at any cost.

```
PS C:\Users\ADMIN\Internship\ob-fe-customer>  git fetch --all

Fetching origin

remote: Enumerating objects: 5, done.

remote: Counting objects: 100% (5/5), done.

remote: Compressing objects: 100% (3/3), done.

remote: Total 3 (delta 1), reused 0 (delta 0), pack-reused o

unpacking objects: 100% (3/3), done.

From https://github. com/ImDwi vedi 1/GitExample2

0ala475. . 828b962 master

->origin/master

PS C:\Users\ADMIN\Internship\ob-fe-customer> Git reset --hard master

HEAD is now at 0al a475 cSs file
```

Git Push

The push term refers to uploading local repository content to a remote repository. Pushing is an act of transfer commits from your local repository to a remote repository. Pushing is capable of overwriting changes; caution should be taken when pushing.Moreover, we can say the push updates the remote refs with local refs. Every time you push into the repository, it is updated with some interesting changes that you made. If we do not specify the location of a repository, then it will push to the default location at origin master.

Git Push Tags

<repository>: The repository is the destination of a push operation. It can be either a URL or the name of a remote repository.

<refspec>: It specifies the destination ref to update source objects.

--all: The word "all" stands for all branches. It pushes all branches.

--prune: It removes the remote branches that do not have a local counterpart. Means, if you have a remote branch, say demo, if this branch does not exist locally, then it will be removed.

--mirror: It is used to mirror the repository to the remote. Updated or Newly created local refs will be pushed to the remote end. It can be force updated on the remote end. The deleted refs will be removed from the remote end.

--dry-run: Dry run tests the commands. It does all this except originally update the repository.

--tags: It pushes all local tags.

--delete: It deletes the specified branch.

Git Push Origin Master

Git push origin master is a special command-line utility that specifies the remote branch and directory. When you have multiple branches and directory, then this command assists you in determining your main branch and repository. Generally, the term origin stands for the remote repository, and master is considered as the main branch. So, the entire statement "git push origin master" pushed the local content on the master branch of the remote location.

```
PS C:\Users\ADMIN\Internship\ob-fe-customer> git push origin master

Enumerating objects: 4, done.

Counting objects : 100% (4/4), done.

Delta compression using up to 2 threads

Compressing objects: 100% (3/3), done.

Writing objects: 100% (3//3), 757. 29 KiB | 6.95 MiB/s, done.

Total 3 (delta 1), reused 0 (delta 0)

remote: Resolving deltas: 100% (1/1), completed with 1 local object.

To https://github.com/ImDwi vedi 1/GitExample2.git

828b962. .Od5191f master ->master
```

Git push -v/--verbose

The -v stands for verbosely. It runs commands verbosely. It pushed the repository and gave a detailed explanation about objects. Suppose we have added a newfile2.txt in our local repository and committed it. Now, when we push it on remote, it will give more description than the default git push.

```
PS C:\Users\ADMIN\Internship\ob-fe-customer> git push -v

Pushing to https://github. com/TmDwi ved i1/GitExample2.git

Enumerating objects: 4, done.

Counting objects: 100% (4/4), done.

Delta compression using up to 2 threads

Compressing objects: 100% (2/2), done.

writing objects: 100% (3/3), 289 bytes | 48. 00 KiB/s, done.

Total 3 (delta 1), reused 0 (delta 0)

POST git-receive-pack (452 bytes)

remote: Resolving deltas: 100% (1/1), completed with 1 local object.

To https://github. com/ImDwi vedi1/GitExamp le2.git

Od5191f..56afce0 master -> master

updating local tracking ref 'refs/remotes/origin/master"
```

Delete a Remote Branch

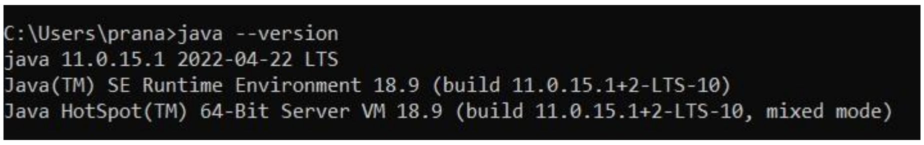
We can delete a remote branch using git push. It allows removing a remote branch from the command line.

```
PS C:\Users\ADMIN\Internship\ob-fe-customer>git push origin --delete edited

To https://github.com/ImDwivedi 1/Gi tExampl e2.git

-[deleted]
```

Q8. Install and Setup Java and Tomcat Setup for Jenkins.

Since Jenkins is written in Java, hence one needs to install Java on the system.

1. **Java:**
   1) Install Java from Oracle website. Can be Java 8 or 11. Follow the steps to setup Java.
   2) Check version of java in command line.



```
C:\Users\prana>java --version
java 11.0.15.1 2022-04-22 LTS
Java(TM) SE Runtime Environment 18.9 (build 11.0.15.1+2-LTS-10)
Java HotSpot(TM) 64-Bit Server VM 18.9 (build 11.0.15.1+2-LTS-10, mixed mode)
```

2. Tomcat:

a. Download the Tomcat binary distribution zip file and unzip the contents
b. Copy Jenkins.war file into the webapps folder of tomcat directory
c. Open terminal and traverse to bin folder in the tomcat folder. type Startup.bat
d. Allow Tomcat in the security permissions.
e. Open browser and type http://localhost:8080/jenkins

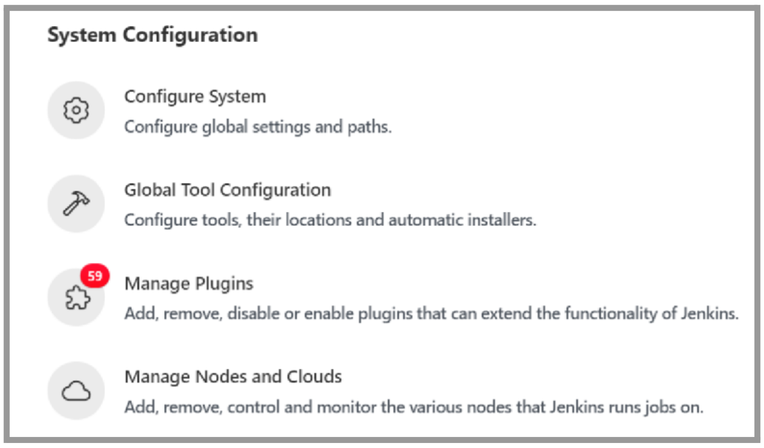
Q9. Integrate GitHub with Jenkins by fetching the source code from GitHub and build it using Jenkins. Jenkins is a CI (Continuous Integration) server and this means that it needs to check out source code from a source code repository and build code. Jenkins has outstanding support for various source code management systems like Subversion, CVS etc.

Github is the fast becoming one of the most popular source code management systems. It is a web based repository of code which plays a major role in DevOps. GitHub provides a common platform for many developers working on the same code or project to upload and retrieve updated code, thereby facilitating continuous integration. Jenkins works with Git through the Git plugin.

Connecting a GitHub private repository to a private instance of Jenkins can be tricky. To do the GitHub setup, make sure that internet connectivity is present in the machine where Jenkins is installed.

Steps:

1. Select Manage Jenkins in the menu in the left
2. Click on Manage Plugins



System Configuration

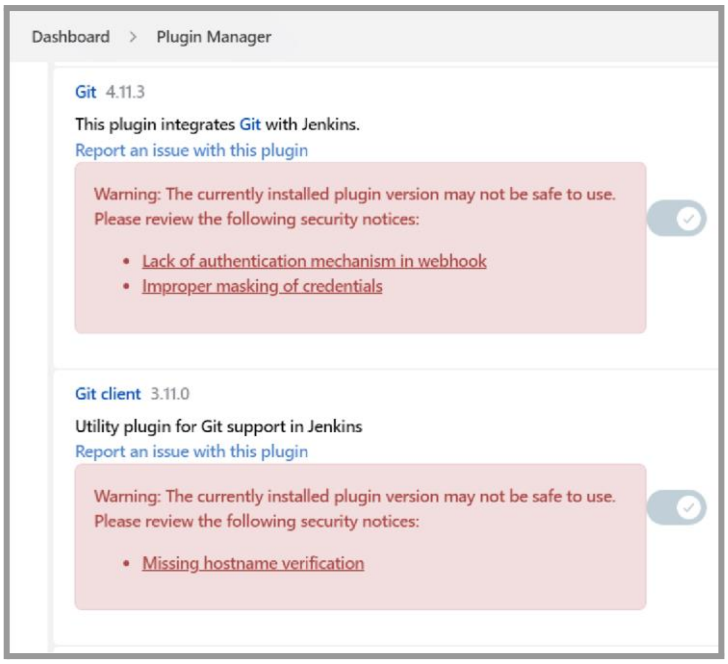
Configure System
Configure global settings and paths.

Global Tool Configuration
Configure tools, their locations and automatic installers.

59
Manage Plugins
Add, remove, disable or enable plugins that can extend the functionality of Jenkins.

Manage Nodes and Clouds
Add, remove, control and monitor the various nodes that Jenkins runs jobs on.

3. Select the Available option, and select Git Plugin and install without restart. 4. In the installed Tab, the git plugin will be installed



Dashboard > Plugin Manager

Git 4.11.3

This plugin integrates Git with Jenkins.

Report an issue with this plugin

Warning: The currently installed plugin version may not be safe to use. Please review the following security notices:

- Lack of authentication mechanism in webhook
- Improper masking of credentials

Git client 3.11.0

Utility plugin for Git support in Jenkins

Report an issue with this plugin

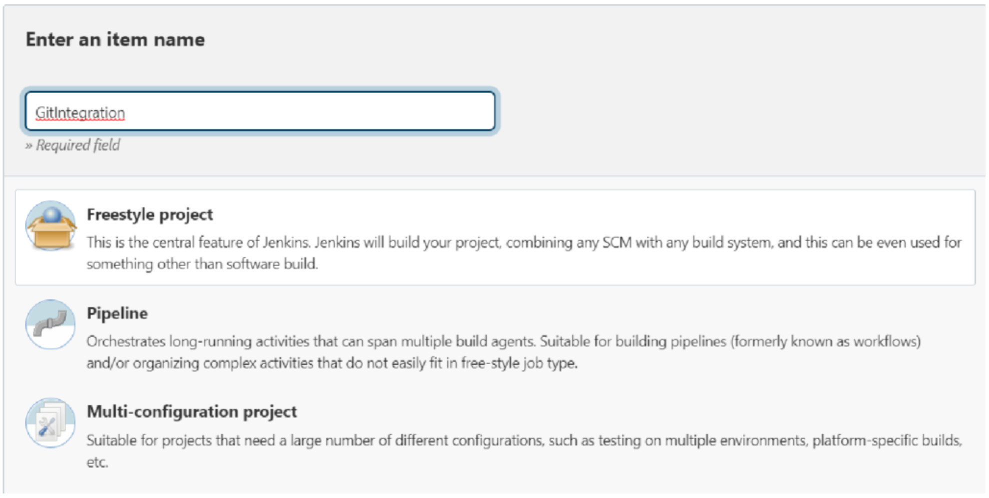
Warning: The currently installed plugin version may not be safe to use. Please review the following security notices:

- Missing hostname verification

5. Restart / Relaunch Jenkins.

To fetch source code and build, the steps are:

1. Create a new job as FreeStyle Project



Enter an item name

GitIntegration

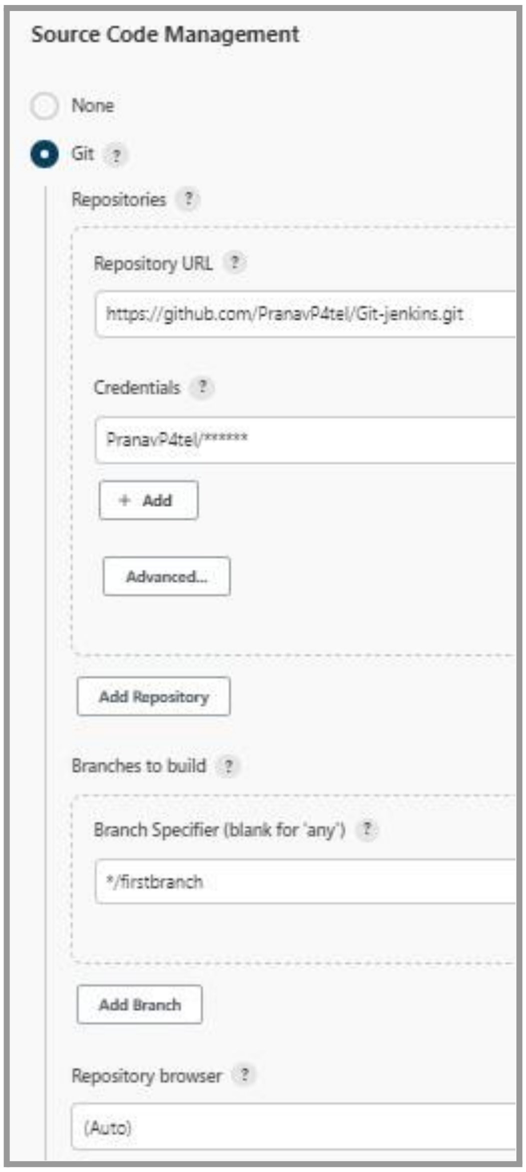
» Required field

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.

2. Enter project information
3. Under Source Code Management, Select Git option.
4. Enter repository URL and add the credentials. Also select branch



Source Code Management

None

Git [✓]

Repositories

Repository URL: https://github.com/PranavP4tel/Git-jenkins.git

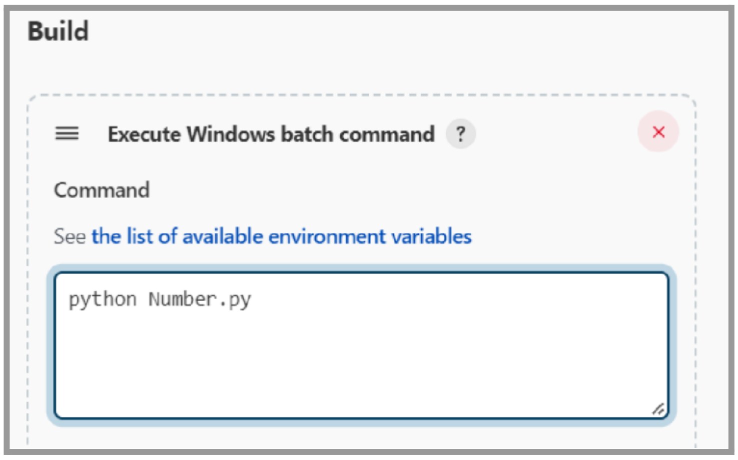
Credentials: PranavP4tel/******

Branches to build

Branch Specifier (blank for 'any'): */firstbranch

Repository browser: (Auto)

5. Add any extra build step



Build

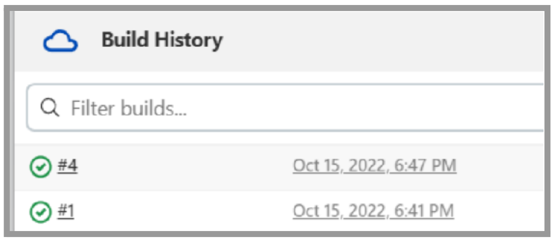
Execute Windows batch command

Command

See the list of available environment variables

```
python Number.py
```

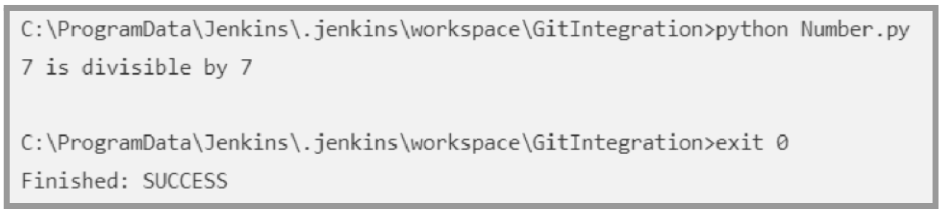
6. Execute project and check everything is working.



Build History

Filter builds...

#4	Oct 15, 2022, 6:47 PM
#1	Oct 15, 2022, 6:41 PM



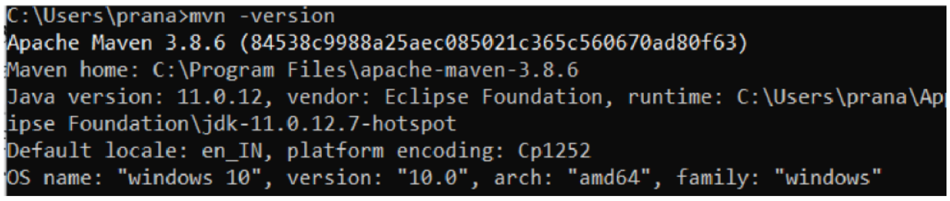
```
C:\ProgramData\Jenkins\.jenkins\workspace\GitIntegration>python Number.py
7 is divisible by 7

C:\ProgramData\Jenkins\.jenkins\workspace\GitIntegration>exit 0
Finished: SUCCESS
```

Q10. Install and set up Maven for build automation.
Maven is a powerful project management and comprehension tool that provides complete build life cycle framework to assist developers. It is based on the concept of a POM (Project Object Model) that includes project information and configuration information for Maven such as construction directory, source directory, test source directory, dependency, Goals, plugins etc.

Maven is build automation tool used basically for Java projects, though it can also be used to build and manage projects written in C#, Scala, Ruby, and other languages. Maven addresses two aspects of building software: 1st it describes how software is build and 2nd it describes its dependencies.

Installing Maven:

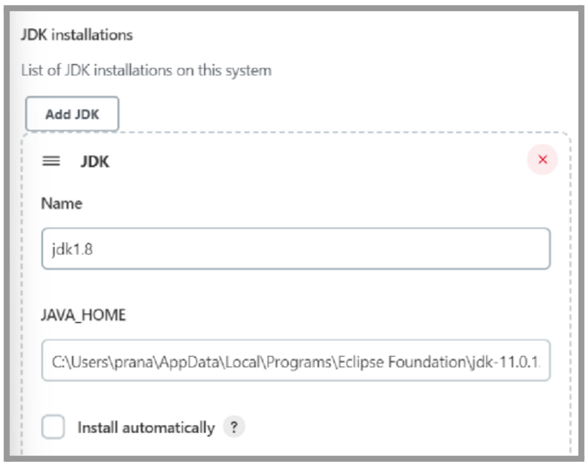
1. Go to apache's official website and download maven zip file.
2. Extract the maven zip.
3. Setup Java Home and Maven Home
4. Check installation by the following



```
C:\Users\prana>mvn -version
Apache Maven 3.8.6 (84538c9988a25aec085021c365c560670ad80f63)
Maven home: C:\Program Files\apache-maven-3.8.6
Java version: 11.0.12, vendor: Eclipse Foundation, runtime: C:\Users\prana\Ap
ipse Foundation\jdk-11.0.12.7-hotspot
Default locale: en_IN, platform encoding: Cp1252
OS name: "windows 10", version: "10.0", arch: "amd64", family: "windows"
```

Install Maven in Jenkins:

1. Click on Manage Jenkins
2. Select Global tool configuration
3. Add a JDK by adding the Name and the Path or select install automatically.



JDK installations

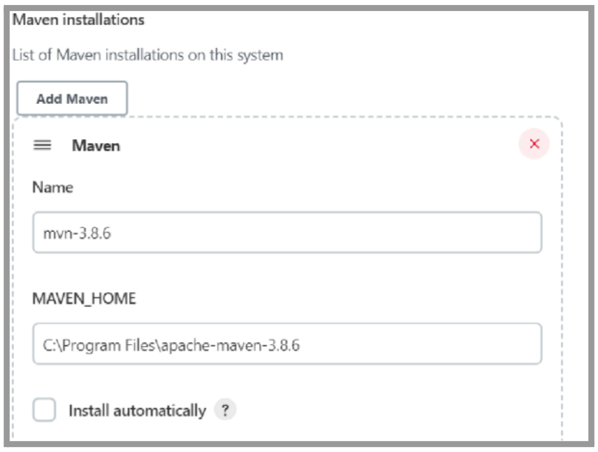
List of JDK installations on this system

Name: jdk1.8

JAVA_HOME: C:\Users\prana\AppData\Local\Programs\Eclipse Foundation\jdk-11.0.1

☐ Install automatically

4. Select Add Maven option and follow the same as above


Maven installations

List of Maven installations on this system

Maven

Name: mvn-3.8.6

MAVEN_HOME: C:\Program Files\apache-maven-3.8.6

Install automatically: ☐

5. Save the changes. Now we can create a job with Maven Project.

Q10. Install and setup Jenkins.
Jenkins is an open source automation tool written in Java programming language that allows continuous integration.

Jenkins builds and tests our software projects which continuously making it easier for developers to integrate changes to the project, and making it easier for users to obtain a fresh build.

It also allows us to continuously deliver our software by integrating with a large number of testing and deployment technologies. Jenkins offers a straightforward way to set up a continuous integration or continuous delivery environment for almost any combination of languages and source code repositories using pipelines, as well as automating other routine development tasks.

With the help of Jenkins, organizations can speed up the software development process through automation. Jenkins adds development life-cycle processes of all kinds, including build, document, test, package, stage, deploy static analysis and much more.

Jenkins achieves CI (Continuous Integration) with the help of plugins. Plugins is used to allow the integration of various DevOps stages. If you want to integrate a particular tool, you have to install the plugins for that tool. For example: Maven 2 Project, Git, HTML Publisher, Amazon EC2, etc.

For example: If any organization is developing a project, then Jenkins will continuously test your project builds and show you the errors in early stages of your development.

Possible steps executed by Jenkins are for example:

- Perform a software build using a build system like Gradle or Maven Apache
- Execute a shell script
- Archive a build result
- Running software tests

Installing Jenkins:
Requirements :

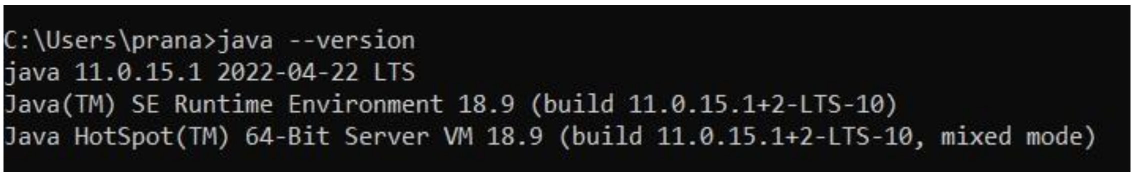
- Only requires a JDK or a JRE.
- Space : Atleast 1 GB
- Ram : Atleast 2GB

Steps :

1. **Install Java**

a. Install Java from Oracle website. Can be Java 8 or 11. Follow the steps to setup Java.

b. Check version of java in command line.



```
C:\Users\prana>java --version
java 11.0.15.1 2022-04-22 LTS
Java(TM) SE Runtime Environment 18.9 (build 11.0.15.1+2-LTS-10)
Java HotSpot(TM) 64-Bit Server VM 18.9 (build 11.0.15.1+2-LTS-10, mixed mode)
```

2. Install Jenkins

a. Download the war file from it's official website

b. Start the installer. Select the default location

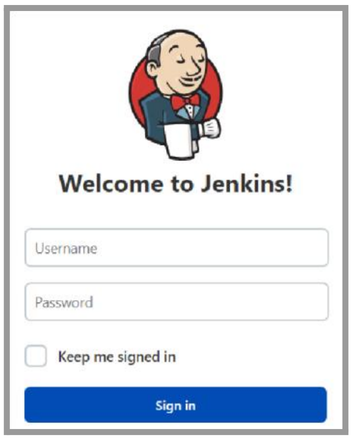
c. Setup the username and password

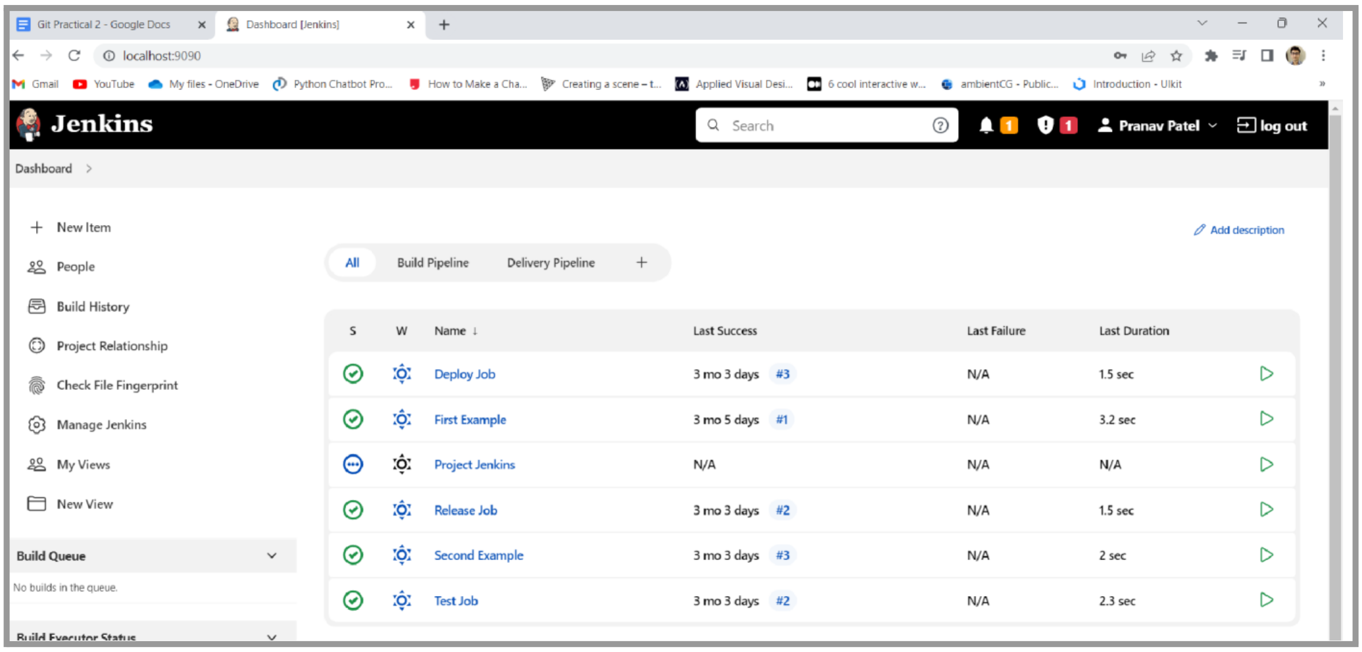
d. Select port number

e. Select directory where java was installed

f. Select features to install and finish installation

g. Jenkins is now downloaded. Access it via http://localhost:9090





S	W	Name ↓	Last Success	Last Failure	Last Duration
✓		Deploy Job	3 mo 3 days #3	N/A	1.5 sec
✓		First Example	3 mo 5 days #1	N/A	3.2 sec
⋯		Project Jenkins	N/A	N/A	N/A
✓		Release Job	3 mo 3 days #2	N/A	1.5 sec
✓		Second Example	3 mo 3 days #3	N/A	2 sec
✓		Test Job	3 mo 3 days #2	N/A	2.3 sec

Q11. Create Jenkins Continuous integration/ Continuous deployment pipeline, Set up the build jobs in the order of code ,build ,test ,deploy .

In Jenkins, a pipeline is a collection of events or jobs which are interlinked with one another in a sequence.

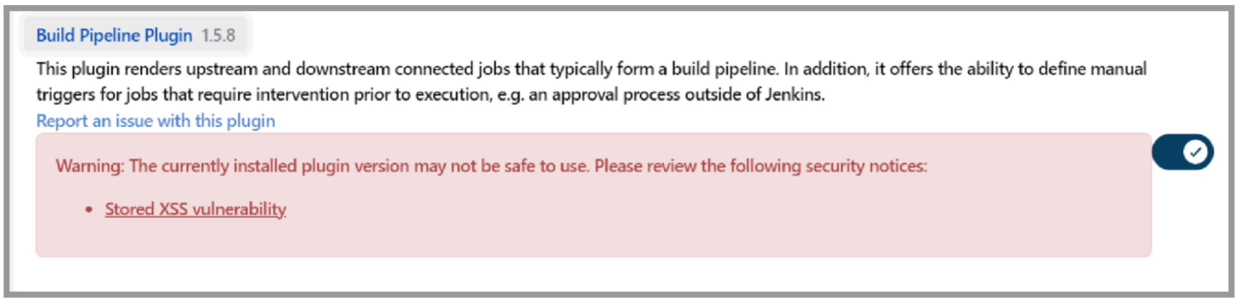
It is a combination of plugins that support the integration and implementation of continuous delivery pipelines using Jenkins.

In other words, a Jenkins Pipeline is a collection of jobs or events that brings the software from version control into the hands of the end users by using automation tools. It is used to incorporate continuous delivery in our software development workflow.

A pipeline has an extensible automation server for creating simple or even complex delivery pipelines "as code", via DSL (Domain-specific language).

To create a CI CD Pipeline in jenkins, we will first execute one task and let the others be executed sequentially.
Steps:

1. Install build pipeline plugin in jenkins.



Build Pipeline Plugin 1.5.8

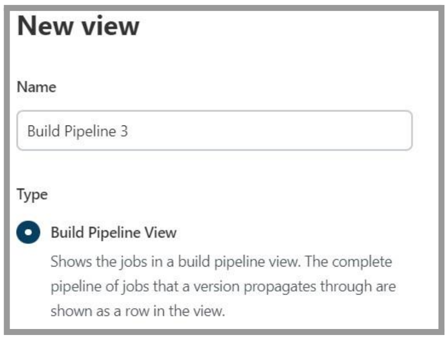
This plugin renders upstream and downstream connected jobs that typically form a build pipeline. In addition, it offers the ability to define manual triggers for jobs that require intervention prior to execution, e.g. an approval process outside of Jenkins.

Report an issue with this plugin

Warning: The currently installed plugin version may not be safe to use. Please review the following security notices:

- Stored XSS vulnerability

2. Create a Build Pipeline View by clicking on the "+" sign in the Dashboard



New view

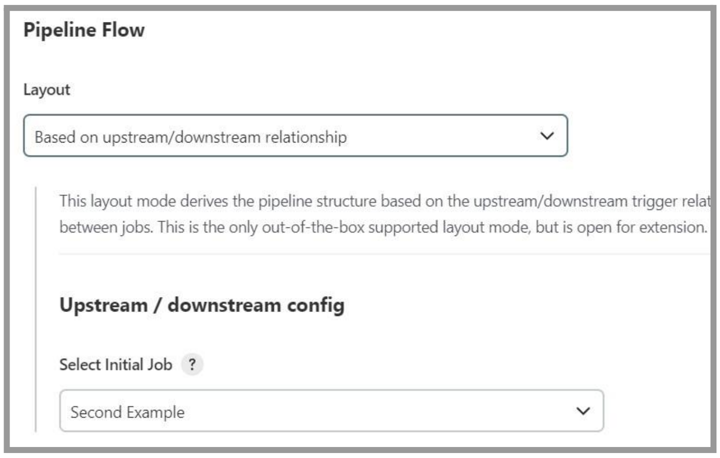
Name

Build Pipeline 3

Type

● Build Pipeline View

Shows the jobs in a build pipeline view. The complete pipeline of jobs that a version propagates through are shown as a row in the view.



Pipeline Flow

Layout

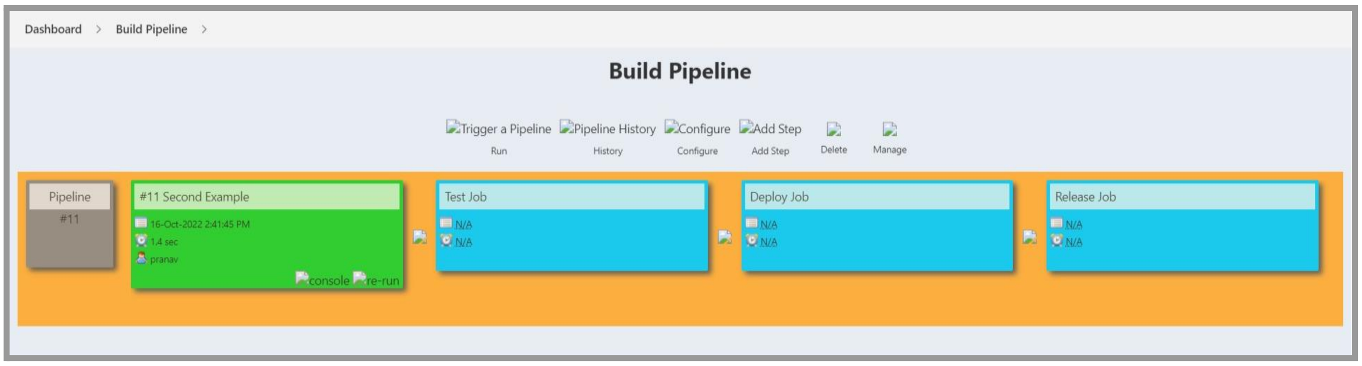
Based on upstream/downstream relationship

This layout mode derives the pipeline structure based on the upstream/downstream trigger relat
between jobs. This is the only out-of-the-box supported layout mode, but is open for extension.

Upstream / downstream config

Select Initial Job ?

Second Example


Dashboard > Build Pipeline >

Build Pipeline

Trigger a Pipeline Run | Pipeline History History | Configure Configure | Add Step Add Step | Delete | Manage

Pipeline #11

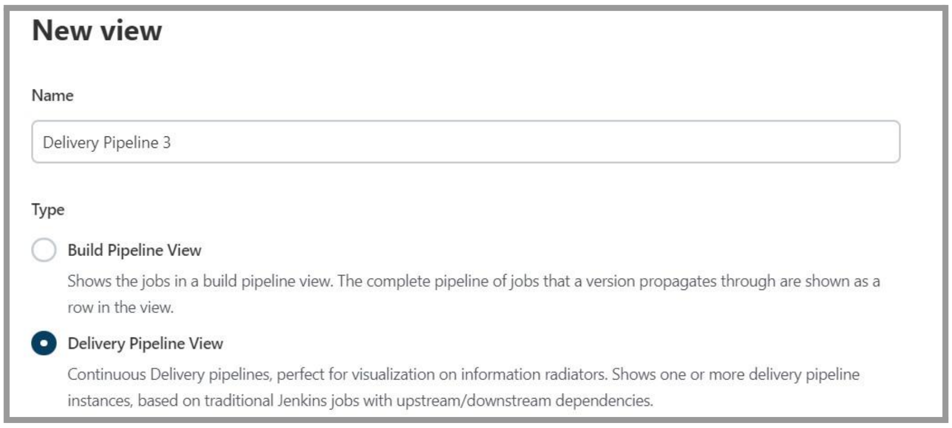
#11 Second Example
16-Oct-2022 2:41:45 PM
1.4 sec
pranav
console re-run

Test Job
N/A
N/A

Deploy Job
N/A
N/A

Release Job
N/A
N/A

3. Create the Delivery Pipeline view in the similar fashion


New view

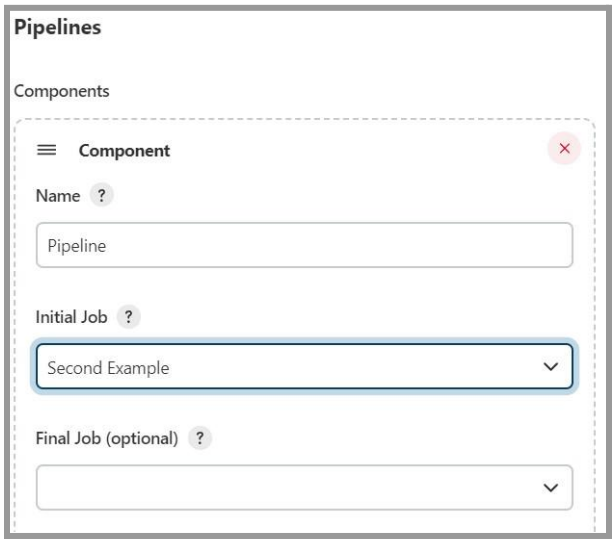
Name

Delivery Pipeline 3

Type

Build Pipeline View
Shows the jobs in a build pipeline view. The complete pipeline of jobs that a version propagates through are shown as a row in the view.

[✓ Delivery Pipeline View]
Continuous Delivery pipelines, perfect for visualization on information radiators. Shows one or more delivery pipeline instances, based on traditional Jenkins jobs with upstream/downstream dependencies.


Pipelines

Components

Component

Name ?

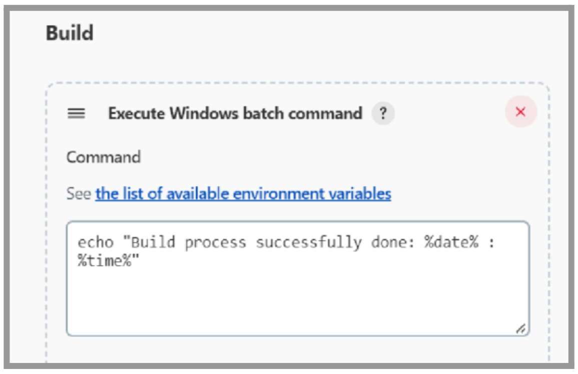
Pipeline

Initial Job ?

Second Example

Final Job (optional) ?

4. Create a Job which will be executed first. (Select No SCM and no Build Triggers). The job will contain a windows batch command to print that the job is completed.


Build

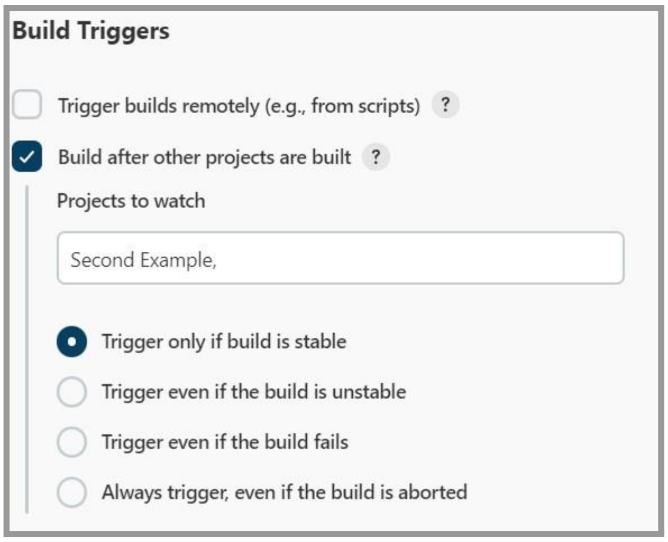
Execute Windows batch command ?

Command

See the list of available environment variables

```
echo "Build process successfully done: %date% :
%time%"
```

5. Create a Test job. No Source code Management is required here. Select a build trigger to build after Job 1 is completed. Also add the windows batch command.

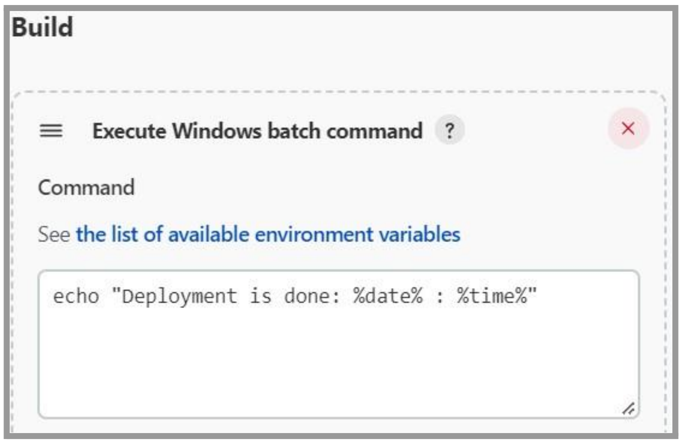

Build Triggers

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

☑ Build after other projects are built ?

Projects to watch: Second Example,

● Trigger only if build is stable
○ Trigger even if the build is unstable
○ Trigger even if the build fails
○ Always trigger, even if the build is aborted


Build

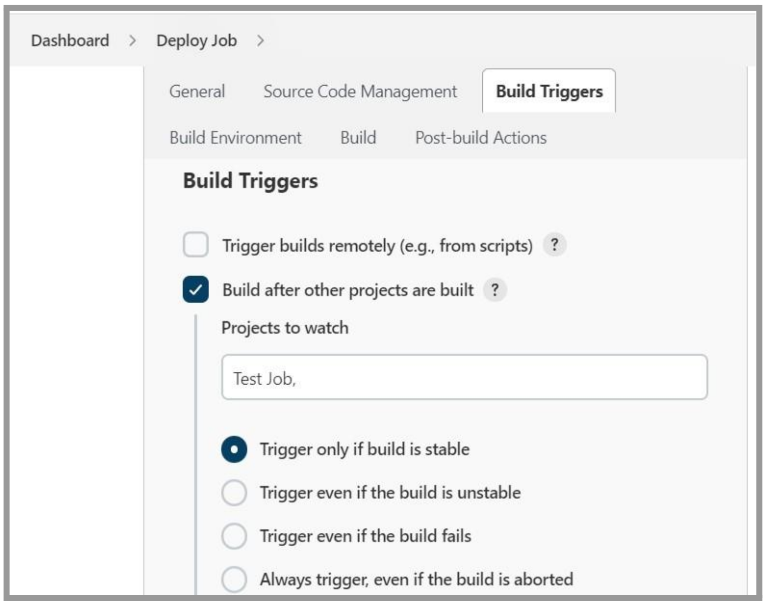
Execute Windows batch command ?

Command

See the list of available environment variables

```
echo "Deployment is done: %date% : %time%"
```

6. Create a deployment job which will be the same as above but it will be build after Job 2.


Dashboard > Deploy Job >

General | Source Code Management | **Build Triggers** | Build Environment | Build | Post-build Actions

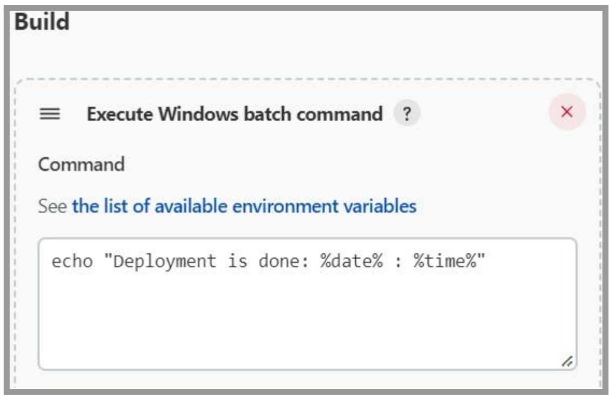
Build Triggers

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

☑ Build after other projects are built ?

Projects to watch: Test Job,

● Trigger only if build is stable
○ Trigger even if the build is unstable
○ Trigger even if the build fails
○ Always trigger, even if the build is aborted


Build

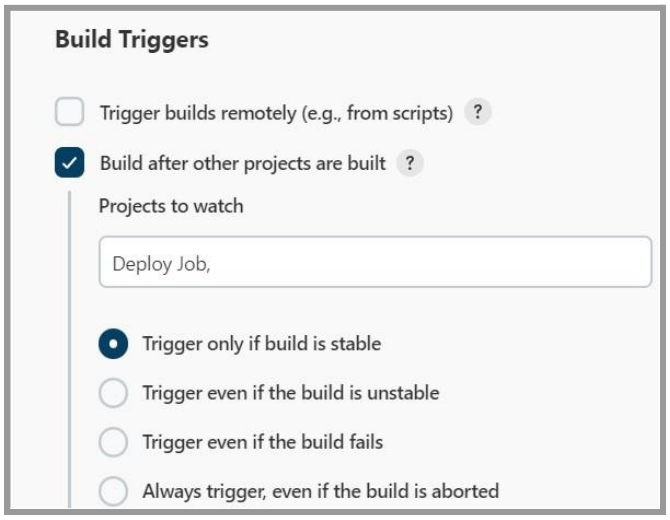
Execute Windows batch command ?

Command

See the list of available environment variables

```
echo "Deployment is done: %date% : %time%"
```

7. Create a release job, same as earlier, which will be built after Job 3.

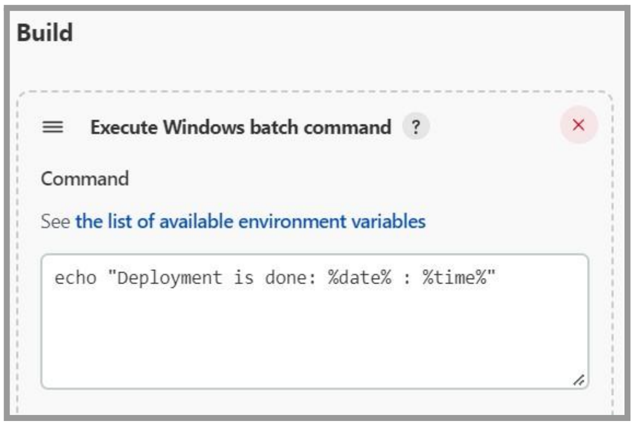

Build Triggers

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

☑ Build after other projects are built ?

Projects to watch: Deploy Job,

● Trigger only if build is stable
○ Trigger even if the build is unstable
○ Trigger even if the build fails
○ Always trigger, even if the build is aborted


Build

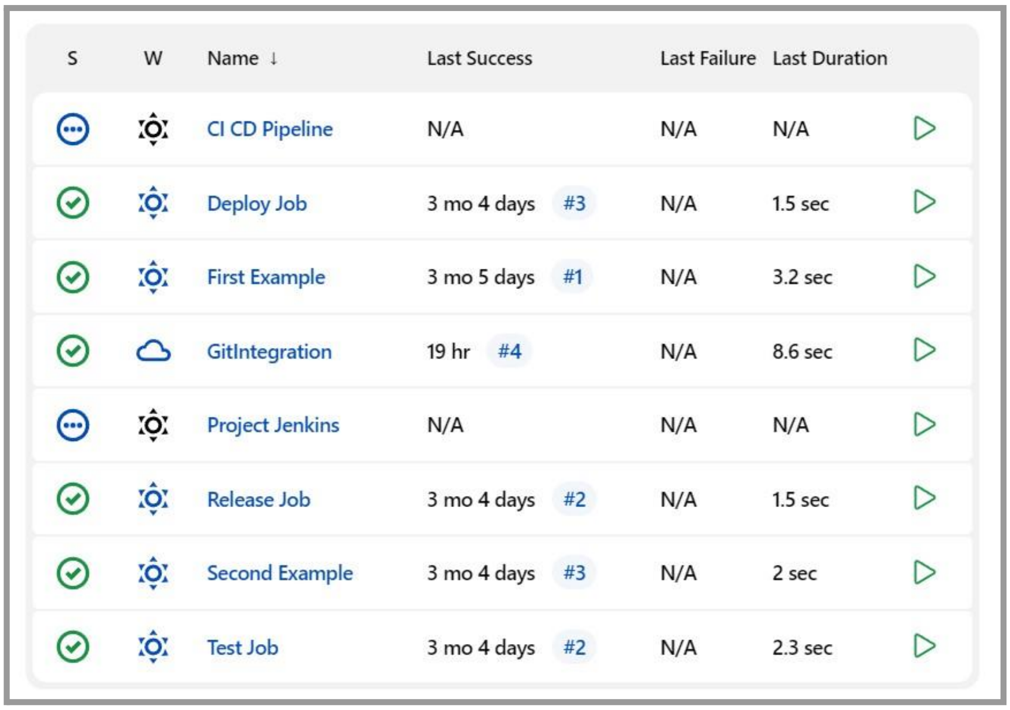
Execute Windows batch command ?

Command

See the list of available environment variables

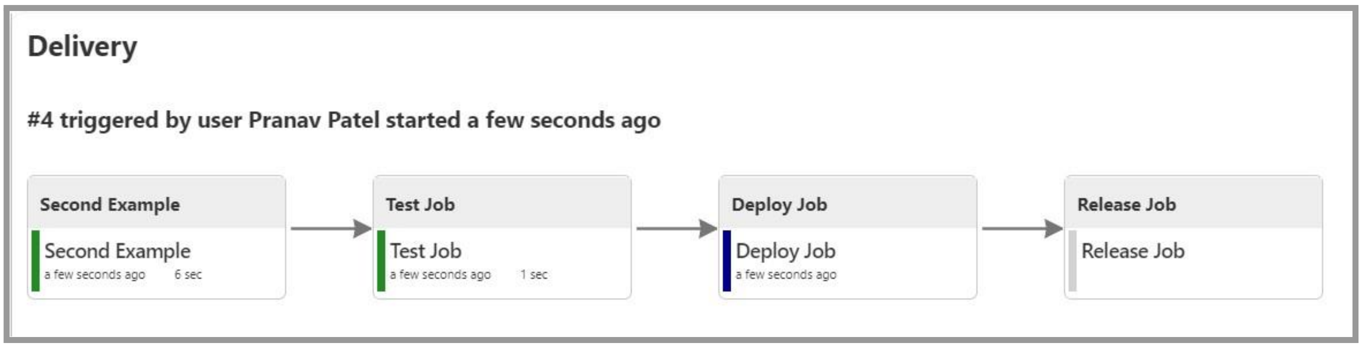
```
echo "Deployment is done: %date% : %time%"
```

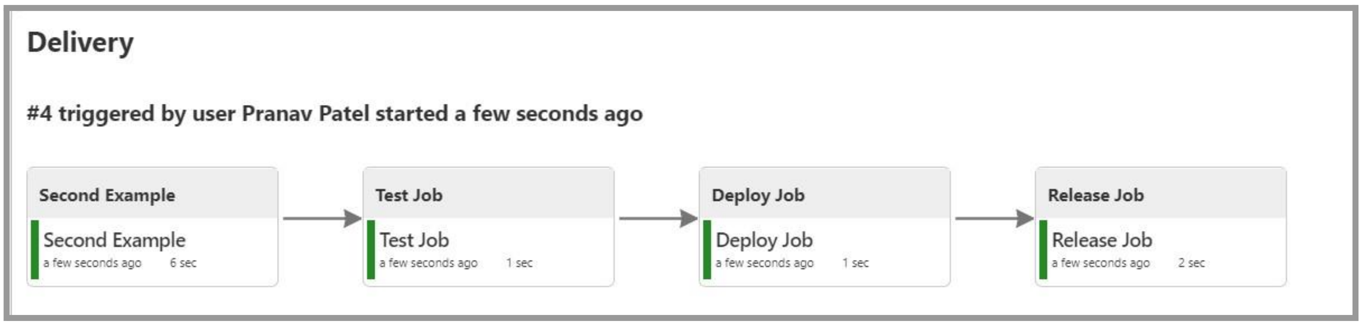
8. Now the pipeline is ready. Execute Job 1 and observe how others get executed



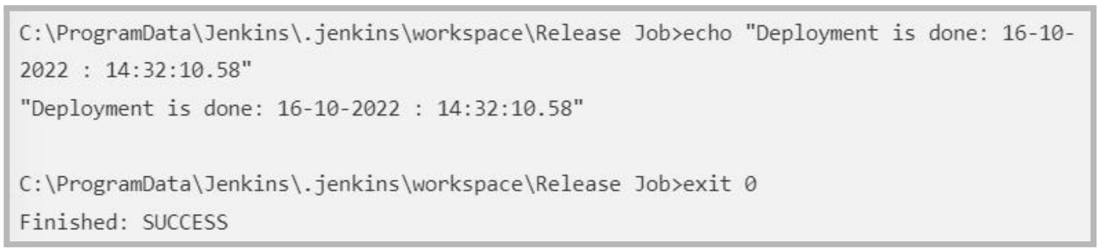
S	W	Name ↓	Last Success	Last Failure	Last Duration	
⋯	☼	CI CD Pipeline	N/A	N/A	N/A	▷
✓	☼	Deploy Job	3 mo 4 days #3	N/A	1.5 sec	▷
✓	☼	First Example	3 mo 5 days #1	N/A	3.2 sec	▷
✓	☁	GitIntegration	19 hr #4	N/A	8.6 sec	▷
⋯	☼	Project Jenkins	N/A	N/A	N/A	▷
✓	☼	Release Job	3 mo 4 days #2	N/A	1.5 sec	▷
✓	☼	Second Example	3 mo 4 days #3	N/A	2 sec	▷
✓	☼	Test Job	3 mo 4 days #2	N/A	2.3 sec	▷

In the delivery pipeline, one can check the status of the tasks. Here each job is build only after the previous has been successfully executed.





Finally, the Console outputs of the tasks can be verified.



```
C:\ProgramData\Jenkins\.jenkins\workspace\Release Job>echo "Deployment is done: 16-10-
2022 : 14:32:10.58"
"Deployment is done: 16-10-2022 : 14:32:10.58"

C:\ProgramData\Jenkins\.jenkins\workspace\Release Job>exit 0
Finished: SUCCESS
```

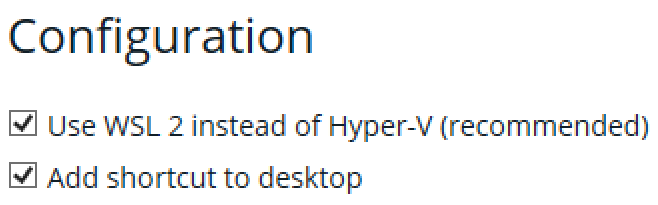
DevOps Practical 12-15: Docker programs	
Name: Hanif Barbhuiya	**Roll number: KCTBCS009**
Date:	**Sign:**

Q12. Install and configure docker

Step 1: Download Docker from https://docs.docker.com/desktop/install/windows-install/

Step 2: Install docker using the installer

Step 3: Configure docker for desktop

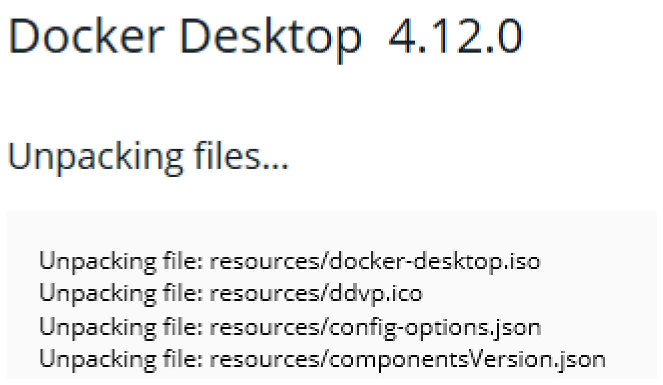


Configuration

☑ Use WSL 2 instead of Hyper-V (recommended)

☑ Add shortcut to desktop

Step 4: Continue installation

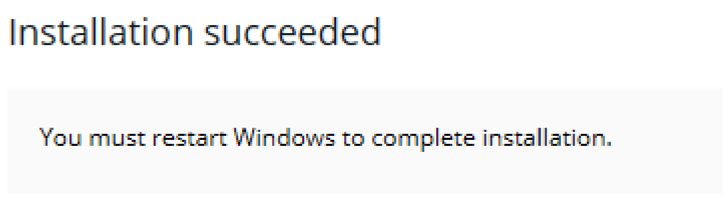


Docker Desktop 4.12.0

Unpacking files...

```
Unpacking file: resources/docker-desktop.iso
Unpacking file: resources/ddvp.ico
Unpacking file: resources/config-options.json
Unpacking file: resources/componentsVersion.json
```

Step 5: Restart your computer



Installation succeeded

You must restart Windows to complete installation.

Step 6: If your admin account is different to your user account, you must add the user to the docker-users group. Run Computer Management as an administrator and navigate to Local Users and Groups > Groups > docker-users. Right-click to add the user to the group. Log out and log back in for the changes to take effect.

Install from the command line

After downloading Docker Desktop Installer.exe, run the following command in a terminal to install Docker Desktop: "Docker Desktop Installer.exe" install

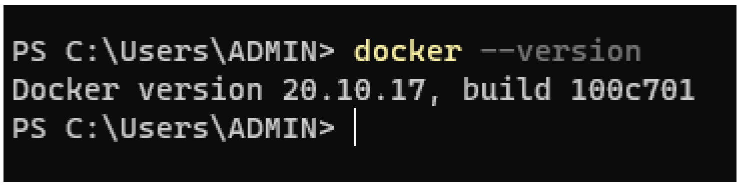
If you're using PowerShell you should run it as:

Start-Process 'Docker Desktop Installer.exe' -Wait install

If using the Windows Command Prompt:

start /w "" "Docker Desktop Installer.exe" install

<u>Check if docker is installed</u>



```
PS C:\Users\ADMIN> docker --version
Docker version 20.10.17, build 100c701
PS C:\Users\ADMIN> |
```

Q13. Create docker image using Dockerfile, Start docker container

Creating DockerFile:

```
PS C:\Users\Admin> cd MyDockerImages

PS C:\Users\Admin\MyDockerImages> New-item Newdockerfile

    Directory: C:\Users\Admin\MyDockerImages

Mode             LastWriteTime       Length Name

----             -------------       ------ ----

-a----       02-08-2022     11:15          0 Newdockerfile


PS C:\Users\Admin\MyDockerImages> Set-Content C:\Users\Admin\MyDockerImages\Newdockerfile

cmdlet Set-Content at command pipeline position 1

Supply values for the following parameters:

Value[0]: FROM UBUNTU:20.04

Value[1]: RUN apt update

Value[2]: RUN apt install default-jdk -y

Value[3]: COPY ..

Value[4]: RUN ["javac","Demo.java"]
```

```
Value[5]: CMD ["java","Demo"]

Value[6]:


PS C:\Users\Admin\MyDockerImages> cat Newdockerfile

FROM UBUNTU:20.04

RUN apt update

RUN apt install default-jdk -y

COPY ..

RUN ["javac","Demo.java"]

CMD ["java","Demo"]
```

<u>Building the Docker File and creating an image</u>

```
PS C:\Users\Admin\MyDockerImages> docker build .

[+] Building 191.4s (7/7) FINISHED

 => [internal] load build definition from Dockerfile                                   0.0s

 => => transferring dockerfile: 31B                                                    0.0s

 => [internal] load .dockerignore                                                      0.0s

 => => transferring context: 2B                                                        0.0s

 => [internal] load metadata for docker.io/library/ubuntu:latest                       4.4s

 => [auth] library/ubuntu:pull token for registry-1.docker.io                          0.0s

 => [1/2] FROM
docker.io/library/ubuntu@sha256:34fea4f31bf187bc915536831fd0afc9d214755bf700b5cdb1336c82516d
154e   10.0s

 => => resolve
docker.io/library/ubuntu@sha256:34fea4f31bf187bc915536831fd0afc9d214755bf700b5cdb1336c82516d
154e    0.0s

 => => sha256:42ba2dfce475de1113d55602d40af18415897167d47c2045ec7b6d9746ff148f 529B / 529B
0.0s

 => => sha256:df5de72bdb3b711aba4eca685b1f42c722cc8a1837ed3fbd548a9282af2d836d 1.46kB /
1.46kB                  0.0s

 => => sha256:d19f32bd9e4106d487f1a703fc2f09c8edadd92db4405d477978e8e466ab290d 30.43MB /
30.43MB                 8.6s

 => => sha256:34fea4f31bf187bc915536831fd0afc9d214755bf700b5cdb1336c82516d154e 1.42kB / 1.42kB
0.0s
```

```
 => => extracting sha256:d19f32bd9e4106d487f1a703fc2f09c8edadd92db4405d477978e8e466ab290d
1.1s
 => [2/2] RUN apt-get update                                              176.7s
 => exporting to image                                                   0.1s
 => => exporting layers                                                  0.1s
 => => writing image sha256:6336f24393cd69d17de3a3a19cd57d1b44ec652dd88273e548b8c6b6ab4c9b63
0.0s
Use 'docker scan' to run Snyk tests against images to find vulnerabilities and learn how to fix them
```

Image created

```
PS C:\Users\Admin\MyDockerImages> docker images
REPOSITORY   TAG       IMAGE ID       CREATED              SIZE
<none>       <none>    6336f24393cd   About a minute ago   113MB
<none>       <none>    2243a5e562a0   3 days ago           97.9MB
```

Running the Image

```
PS C:\Users\Admin\MyDockerImages> docker run 6336f24393cd
Hello World
```

Pulling the Ubuntu Image

```
PS C:\Users\Admin\MyDockerImages> docker pull ubuntu
Using default tag: latest
latest: Pulling from library/ubuntu
d19f32bd9e41: Already exists
Digest: sha256:34fea4f31bf187bc915536831fd0afc9d214755bf700b5cdb1336c82516d154e
Status: Downloaded newer image for ubuntu:latest
docker.io/library/ubuntu:latest
```

Running the container

```
PS C:\Users\Admin\MyDockerImages> docker run -it -d ubuntu
930ffc6736b2a9c53c66f0676b3fab46c6f5617f35a3fb57495f6139b786f2d9
```

```
PS C:\Users\Admin\MyDockerImages> docker ps -a
```

```
CONTAINER ID   IMAGE          COMMAND              CREATED          STATUS                 PORTS     NAMES
930ffc6736b2   ubuntu         "bash"               15 seconds ago   Up 14 seconds
mystifying_elgamal
280d1270e43a   6336f24393cd   "echo 'Hello World'"   6 minutes ago    Exited (0) 6 minutes ago
flamboyant_bose


PS C:\Users\Admin\MyDockerImages> docker stop 280d1270e43a
280d1270e43a


PS C:\Users\Admin\MyDockerImages> docker start 280d1270e43a
280d1270e43a
```

Q14. Connect to docker container, Copy the website code to the container.

```
PS C:\Users\Admin\MyDockerImages> docker exec -it 930ffc6736b2 bash
root@930ffc6736b2:/# echo hello
hello
root@930ffc6736b2:/# exit
exit
```

Q15. Use docker management commands to

<u>List the images</u>

```
PS C:\Users\Admin\MyDockerImages> docker images
REPOSITORY   TAG       IMAGE ID       CREATED          SIZE
<none>       <none>    6336f24393cd   15 minutes ago   113MB
ubuntu       latest    df5de72bdb3b   5 hours ago      77.8MB
<none>       <none>    2243a5e562a0   3 days ago       97.9MB
```

<u>List the containers</u>

```
PS C:\Users\Admin\MyDockerImages> docker ps -a
CONTAINER ID   IMAGE          COMMAND              CREATED          STATUS                 PORTS     NAMES
930ffc6736b2   ubuntu         "bash"               15 seconds ago   Up 14 seconds
mystifying_elgamal
280d1270e43a   6336f24393cd   "echo 'Hello World'"   6 minutes ago    Exited (0) 6 minutes ago
flamboyant_bose
```

Start and stop container

```
PS C:\Users\Admin\MyDockerImages> docker start 280d1270e43a

280d1270e43a

PS C:\Users\Admin\MyDockerImages> docker stop 280d1270e43a

280d1270e43a
```

Remove container and image

```
PS C:\Users\Admin\MyDockerImages> docker rm 930ffc6736b2

930ffc6736b2

PS C:\Users\Admin\MyDockerImages> docker rmi  df5de72bdb3b

Untagged: ubuntu:latest

Untagged: ubuntu@sha256:34fea4f31bf187bc915536831fd0afc9d214755bf700b5cdb1336c82516d154e

Deleted: sha256:df5de72bdb3b711aba4eca685b1f42c722cc8a1837ed3fbd548a9282af2d836d

PS C:\Users\Admin\MyDockerImages> docker container ls

CONTAINER ID   IMAGE    COMMAND   CREATED   STATUS    PORTS    NAMES
```

DevOps Practical 17-20: Puppet and Kubernetes programs	
Name: Hanif Barbhuiya	**Roll number: KCTBCS009**
Date:	**Sign:**

Configuration Management

Q17. Software Configuration Management and provisioning tools using Puppet.

Configuration management occurs when a configuration platform is used to automate, monitor, design and manage otherwise manual configuration processes. System-wide changes take place across servers and networks, storage, applications, and other managed systems.

An important function of configuration management is defining the state of each system. By orchestrating these processes with a platform, organizations can ensure consistency across integrated systems and increase efficiency. The result is that businesses can scale more readily without hiring additional IT management staff. Companies that otherwise wouldn't have the resources can grow by deploying a DevOps approach.

Configuration management is closely associated with change management, and as a result, the two terms are sometimes confused. Configuration management is most readily described as the automation, management, and maintenance of configurations at each state, while change management is the process by which configurations are redefined and changed to meet the conditions of new needs and dynamic circumstances.

A number of tools are available for those seeking to implement configuration management in their organizations. Puppet has carried the torch in pioneering configuration management, but other companies like Chef and Red Hat also offer intriguing suites of products to enhance configuration management processes. Proper configuration management is at the core of continuous testing and delivery, two key benefits of DevOps.

Components: Configuration Management in DevOps

Based on what we've discussed, you may have already gleaned that configuration management takes on the primary responsibility for three broad categories required for DevOps transformation: identification, control, and audit processes.

Identification:
The process of finding and cataloging system-wide configuration needs.

Control:
During configuration control, we see the importance of change management at work. It's highly likely that configuration needs will change over time, and configuration control allows this to happen in a controlled way as to not destabilize integrations and existing infrastructure.

Audit:
Like most audit processes, a configuration audit is a review of the existing systems to ensure that it stands up to compliance regulation and validations.

Like DevOps, configuration management is spread across both operational and development buckets within an organization. This is by design. There are primary components that go into the comprehensive configuration management required for DevOps:

- Artifact repository
- Source code repository
- Configuration management data architecture

What is Puppet?

- Puppet is a **DevOps configuration management tool**. This is developed by Puppet Labs and is available for both open-source and enterprise versions. It is used to centralize and automate the procedure of configuration management.
- This tool is developed using Ruby DSL (domain-specific language), which allows you to change a complete infrastructure in code format and can be easily managed and configured.
- Puppet tool deploys, configures, and manages the servers. This is used particularly for the automation of hybrid infrastructure delivery and management.
- With the help of automation, Puppet enables system administrators to operate easier and faster.
- Puppet can also be used as a deployment tool as it can deploy software on the system automatically. Puppet implements infrastructure as a code, which means that you can test the environment for accurate deployment.
- Puppet supports many platforms such as Microsoft Windows, Debian/Ubuntu, Red Hat/CentOS/Fedora, MacOS X, etc.
- Puppet uses the client-server paradigm, where one system in any cluster works as the server, called the puppet master, and other works as a client on nodes called a slave.

Puppet Blocks

Puppet provides the flexibility to integrate Reports with third-party tools using Puppet APIs.

Four types of Puppet building blocks are

1. Resources
2. Classes
3. Manifest
4. Modules

Puppet Resources:

Puppet Resources are the building blocks of Puppet.

Resources are the inbuilt functions that run at the back end to perform the required operations in puppet.

Puppet Classes:

A combination of different resources can be grouped together into a single unit called class.

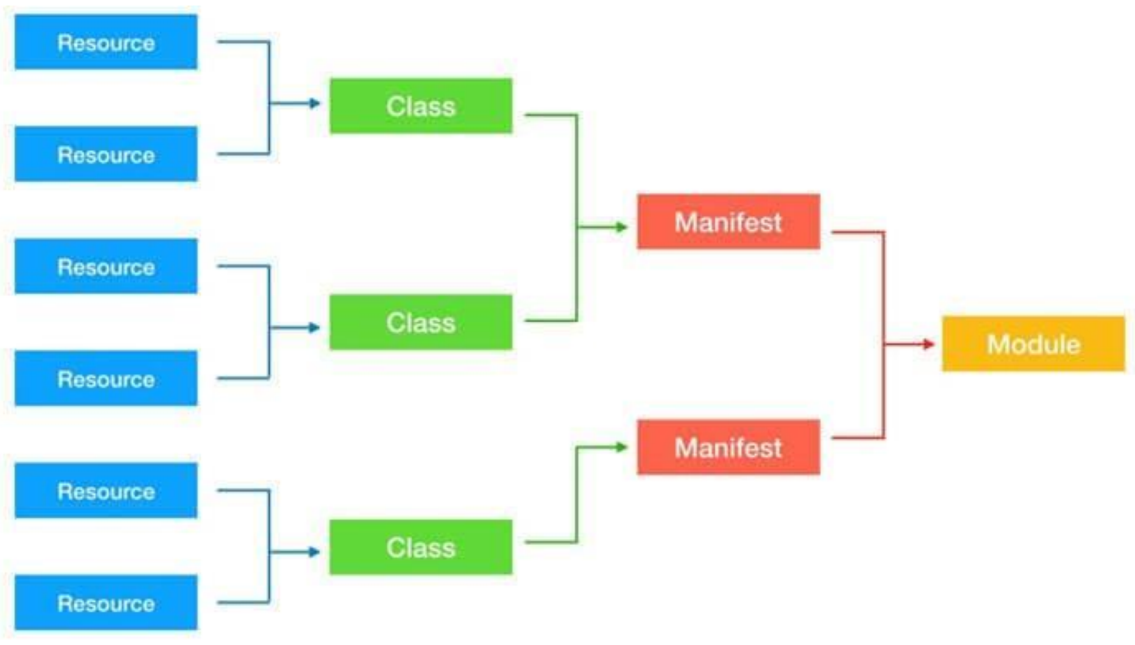
Puppet Manifest:

Manifest is a directory containing puppet DSL files. Those files have a .pp extension. The .pp extension stands for puppet program. The puppet code consists of definitions or declarations of Puppet Classes.

Puppet Modules:

Modules are a collection of files and directories such as Manifests, Class definitions. They are the re-usable and sharable units in Puppet.

For example, the MySQL module to install and configure MySQL or the Jenkins module to manage Jenkins, etc..



Windows nodes cannot serve as puppet master servers.

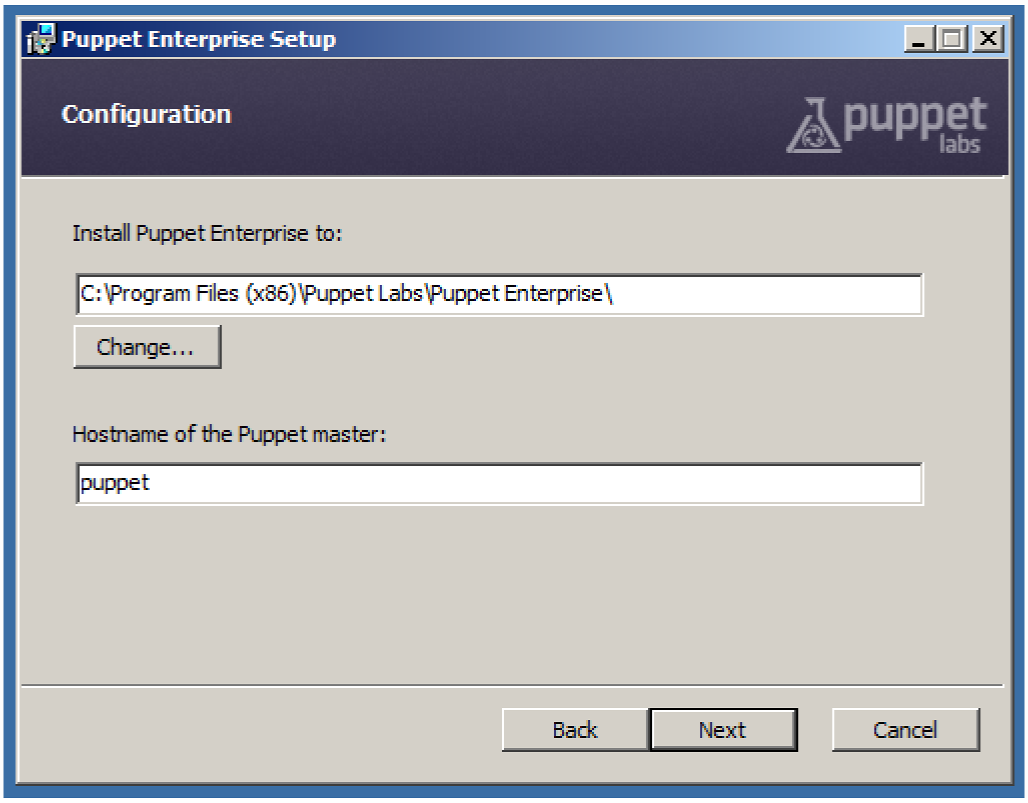
- If your Windows nodes will be fetching configurations from a puppet master, you will need a *nix server to run as puppet master at your site.

- If your Windows nodes will be compiling and applying configurations locally with puppet apply, you should disable the puppet agent service on them after installing Puppet.

Installing Puppet

To install Puppet, simply download and run the installer, which is a standard Windows .msi package and will run as a graphical wizard.

The installer must be run with elevated privileges. Installing Puppet **does not** require a system reboot.
The only information you need to specify during installation is **the hostname of your puppet master server:** (If you are using puppet apply for node configuration instead of a puppet master, you can just enter some dummy text here.)
Note that you can download and install Puppet Enterprise on up to ten nodes at no charge. No licence key is needed to run PE on up to ten nodes.


Puppet Enterprise Setup

Configuration

Install Puppet Enterprise to:

C:\Program Files (x86)\Puppet Labs\Puppet Enterprise\

Hostname of the Puppet master:

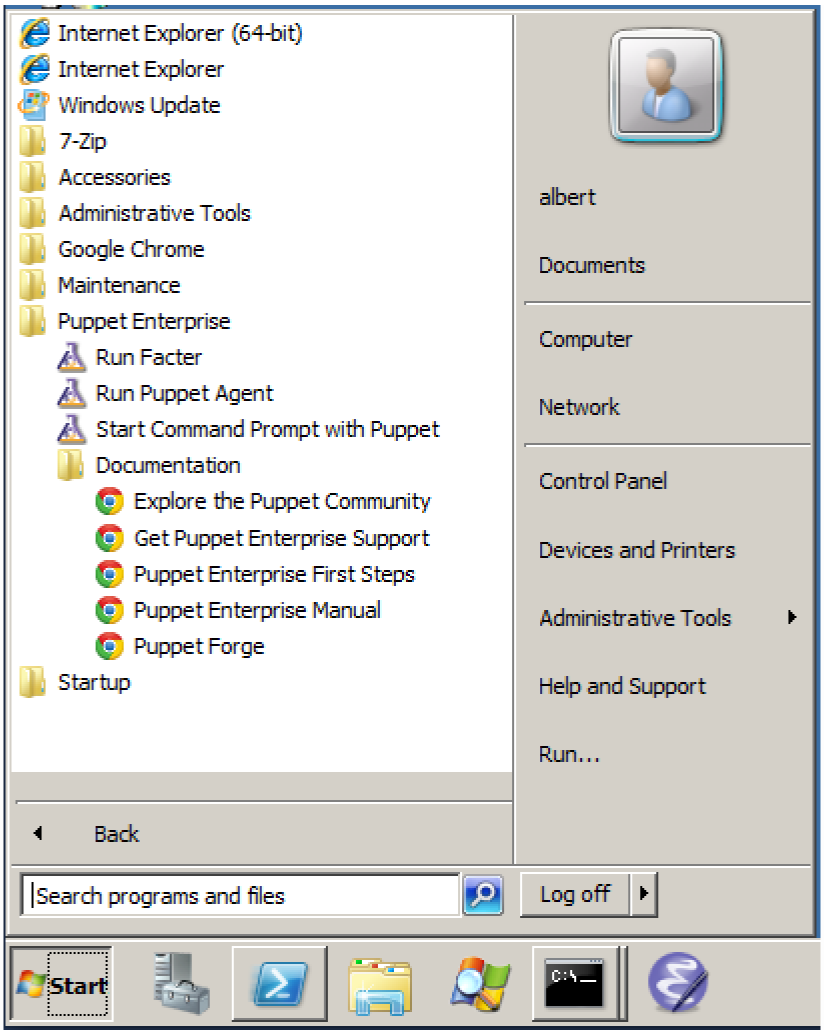
puppet

After Installation

Once the installer finishes:

- Puppet agent will be running as a Windows service, and will fetch and apply configurations every 30 minutes. You can now assign classes to the node on your puppet master or console server. Puppet agent can be started and stopped with the Service Control Manager or the sc.exe utility; see Running Puppet on Windows for more details.

- The Start menu will contain a Puppet folder, with shortcuts for running puppet agent manually, for running Facter, and for opening a command prompt for use with the Puppet tools. See Running Puppet on Windows for more details. The Start menu folder also contains documentation links.



Puppet Coding Style

In Puppet, the coding style describes all the requirements that must be followed when attempting to transform infrastructure on the system configuration into code. Puppet requires resources to work and execute all of its defined tasks.

As we know, the puppet employs Ruby language as its encoding language, which provides several predefined features, and with the help of these features, it is very easy to complete the things with the simple configuration on code.

Fundamental Units

There are many fundamental coding styles used by a puppet that is easy to understand and use. Let's see some of them:

Resources

In puppet, resources are the basic unit used for modeling the system configurations.

Resources are the building blocks of a puppet. Each resource describes the desired state for some aspects of a system, such as service, file, and package.

Resources are the predefined functions that allow the users or developers to develop custom resources, with the help of which we can manage any particular unit of a system.

Resources in the puppet are aggregated together by using either "define" or "classes." This feature provides help in organizing a module

Every resource declaration at least contains a resource type, a title, and a set of attributes.

Syntax:

```
<TYPE> { '<TITLE>':

<ATTRIBUTE> => <VALUE>, }
```

Let's see one sample resource where a title and a list of attributes are defined. Each resource contain a default value which could be overridden as per our requirement.

```
file {

  '/etc/passwd':

  owner => superuser,

  group => superuser,

  mode => 644,

}
```

The above command specifies the permission for a particular file.

Each time the command is executed on any system, it will validate that the system's passwd file is configured as described. Here, the file defined before the colon (:) is the title of the resource, which we can use as a resource in other puppet configuration section.

Let's define local name in addition to the title:

```
file { 'sshdconfig':

  name => $operaSystem ? {

    solaris => '/usr/local/etc/ssh/sshd_config',

    default => '/etc/ssh/sshd_config',

  },

  owner => superuser,

  group => superuser,

  mode => 644,

}
```

It is very convenient to refer to file resources in the configuration by using the title, which is always the same, which helps to reduce the repetition of OS-related logic.

Example of a user resource declaration:

```
user { 'Nik':
 ensure     => present,
 uid       => '100',
 gid       => '100',
 shell      => '/bin/bash',
 home       => '/home/Nik'
}
```

Another example may be using a service which is dependent on a file:

```
service { 'sshd':
   subscribe => File[sshdconfig],
}
```

Here, the sshd service will always restart once the sshdconfig file changes. The important thing is File[sshdconfig] is a declaration as file as in the lower case, but if we modify it to FILE[sshdconfig], then it would have been a reference.

The main point to be noted is that we can declare a resource only once per Config file. If we repeat the declaration of the same resource, then it will cause an error.

Even we can manage multiple relationships through the resource dependency:

```
service { 'sshd':
   require => File['sshdconfig', 'sshconfig', 'authorized_keys']
}
```

Puppet Manifest

In puppet, all the programs are written in Ruby programming language and added with an extension of .pp is known as manifests. The full form of .pp is the puppet program.

Manifest files are puppet programs. This is used to manage the target host system. All the puppet programs follow the puppet coding style.

We can use a set of different kinds of resources in any manifest, which is grouped by definition and class.

Puppet manifest also supports the conditional statement.

Manifest Components

Puppet manifest has the following components:

- **Files:** Files are the plain text files that can be directly deployed on your puppet clients. Such as yum.conf, httpd.con, etc.

- **Resources:** Resources are the elements that we need to evaluate or change. Resources can be packages, files, etc.
- **Templates:** This is used to create configuration files on nodes and which we can reuse later.
- **Nodes:** Block of code where all the information and definition related to the client are defined here.
- **Classes:** Classes are used to group different types of resources.

Writing Manifests

Working with Variables

Puppet provides many in-built variables that we can use in the manifest. As well as we can create our own variable to define in puppet manifest. Puppet provides different types of variables. Some frequently used variables are strings or an array of string.

Let's see an example for string variable:

```
$package = "vim"

package {  $package:

  ensure => "installed"

}
```

Working with Loops

Loops are used to run the same set of code multiple times until a defined condition becomes true. To perform the loop, we can use an array. Let's see an example:

```
$packages = ['vim', 'git', 'curl']

package { $packages:

  ensure => "installed"

}
```

Using Conditionals

Puppet allows us to use different types of conditional statements. Such as if-else statement, case statements, etc. Let's see an example:

```
if $Color != 'White' {

  warning('This color is not good for wall')

} else {

  notify { 'This color is best for wall': }

}
```

Example

Writing a Manifest

As we know, we can create our resources. Let's start with common resources like notify resources.

```
notify { 'greeting':
  message => 'Hello, world!'
}
```

In the above code, the **notify** is the resource, and the message is the attribute. The **message** has attributes that are separated from their values by a comma, which is a very general way of identifying key-value pairs in Ruby, PHP, Perl, and other scripting languages.

Applying a Manifest

The main quality of the puppet is the ease of testing your code. In this, to work on puppet manifests, there is no need to configure complicated testing environments.

To apply the manifest, puppet uses apply command, which tells puppet to apply a single puppet manifest:

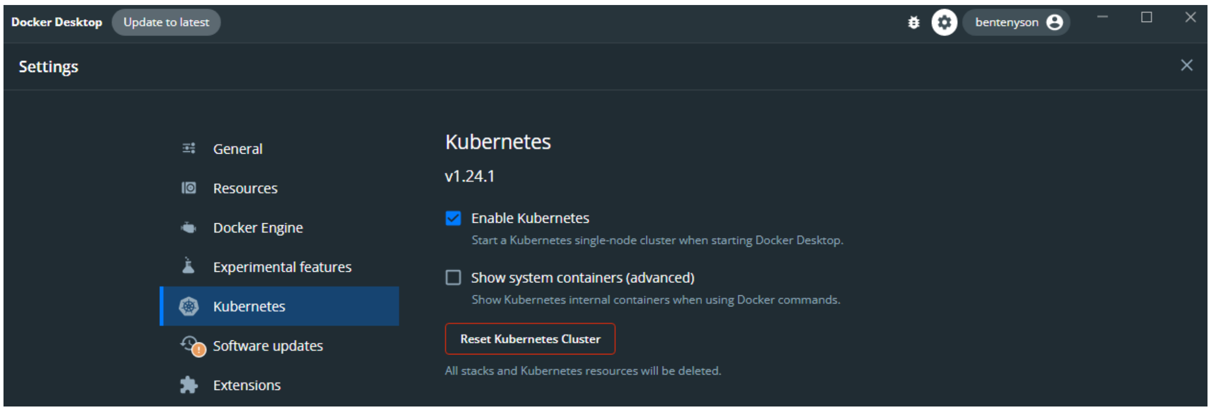
```
# puppet apply helloworld.pp
Notice: Compiled catalog for puppetclient in environment production in 0.03 seconds
Notice: Hello, World!
Notice: /Stage[main]/Main/Notify[greeting]/message:
  defined 'message' as 'Hello, World!'
Notice: Applied catalog run in 0.01 seconds
```

Q18. Configure Kubernetes, Configure Kubernetes Dashboard.

Dashboard is a web-based Kubernetes user interface. We can use Dashboard to deploy and troubleshoot containerized applications to a Kubernetes cluster, and to manage the cluster resources. We can also use Dashboard to get an overview of applications running on our cluster, as well as for creating or modifying individual Kubernetes resources (such as Deployments, Jobs, DaemonSets, etc). For example, we can scale a Deployment, initiate a rolling update, restart a pod, etc.

Dashboard also provides information on the state of Kubernetes resources in your cluster and on any errors that may have occurred.

To start with the dashboard creation process, we have to enable the kubernetes feature in docker and ensure that it is running smoothly.

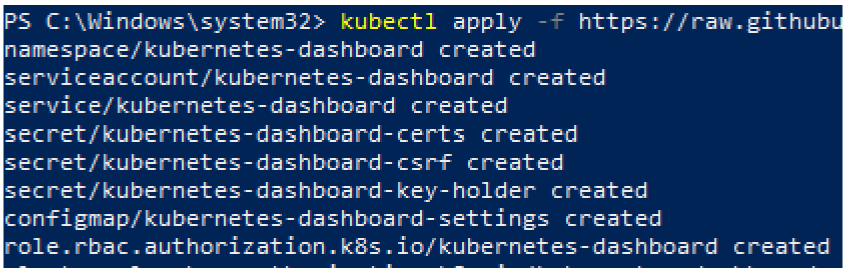


Kubernetes
v1.24.1
[✓] Enable Kubernetes
Start a Kubernetes single-node cluster when starting Docker Desktop.
[ ] Show system containers (advanced)
Show Kubernetes internal containers when using Docker commands.
All stacks and Kubernetes resources will be deleted.

2: Then we run the following command which will create the required components for the dashboard

kubectl apply -f
https://raw.githubusercontent.com/kubernetes/dashboard/v2.6.1/aio/deploy/recommended.yaml

This kubectl command is basically referring to the recommended.yml manifest file stored in the github page for kubernetes dashboard and performing the operations



```
PS C:\Windows\system32> kubectl apply -f https://raw.githubu
namespace/kubernetes-dashboard created
serviceaccount/kubernetes-dashboard created
service/kubernetes-dashboard created
secret/kubernetes-dashboard-certs created
secret/kubernetes-dashboard-csrf created
secret/kubernetes-dashboard-key-holder created
configmap/kubernetes-dashboard-settings created
role.rbac.authorization.k8s.io/kubernetes-dashboard created
```

To protect your cluster data, Dashboard deploys with a minimal access control configuration by default. i.e. is why Currently, Dashboard will only support logging in with a Bearer Token.

For this, we will have to create a new user using the Service Account mechanism of Kubernetes, grant this user admin permissions and login to Dashboard using a bearer token tied to this user.

We will create a new manifest file named dashboard-adminuser.yaml and use it to perform the mentioned operations.

Creating a Service Account

We are creating Service Account with the name admin-user in namespace kubernetes-dashboard first.

File content:

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: admin-user
  namespace: kubernetes-dashboard
```

The we use the kubectl command to execute the operations from the manifest file

```
kubectl apply -f dashboard-adminuser.yaml
```

```
PS C:\Users\shahk\OneDrive\Desktop\Assignments\Year 3\DevOps> kubectl apply -f dashboard-adminuser.yaml
```

Output: serviceaccount/admin-user created

Creating a ClusterRoleBinding

In most cases after provisioning the cluster the ClusterRole cluster-admin already exists in the cluster. We can use it and create only a ClusterRoleBinding for our ServiceAccount. If it does not exist, then we need to create this role first and grant required privileges manually.

File content:

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
```

```
metadata:
  name: admin-user
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: cluster-admin
subjects:
- kind: ServiceAccount
  name: admin-user
  namespace: kubernetes-dashboard
```

The, again using the same kubectl command

kubectl apply -f dashboard-adminuser.yaml apply these changes

```
PS C:\Users\shahk\OneDrive\Desktop\Assignments\Year 3\DevOps> kubectl apply -f dashboard-adminuser.yaml
```

Output: clusterrolebinding.rbac.authorization.k8s.io/admin-user created

Getting a Bearer Token

Now we need to find the token we can use to log in. For it we Execute the following command:

```
kubectl -n kubernetes-dashboard create token admin-user
```

It should print something like:

```
eyJhbGciOiJSUzI1NiIsImtpZCI6Imh2VVdQWmlXMVQ5SS1UelRyREVOLXV0Rks4T3lVODZHRkJ1Z
DNOd2tlaG8ifQ.eyJhdWQiOlsiaHR0cHM6Ly9rdWJlcm5ldGVzLmRlZmF1bHQuc3ZjLmNsdXN0ZXIub
G9jYWwiXSwiZXhwIjoxNjYyOTA1NjkwLCJpYXQiOjE2NjI5MDIwOTAsImlzcyI6Imh0dHBzOi8va3Vi
ZXJuZXRlcy5kZWZhdWx0LnN2Yy5jbHVzdGVyLmxvY2FsIiwia3ViZXJuZXRlcy5pbyI6eyJuYW1lc3Bh
Y2UiOiJrdWJlcm5ldGVzLWRhc2hib2FyZCIsInNlcnZpY2VhY2NvdW50Ijp7Im5hbWUiOiJhZG1pbi11c2
VyIiwidWlkIjoiNDYxMzZmMzYtYWViZi00NmRlLWJhYjItMmYwYWI4MzY1NWFkIn19LCJuYmYiOj
E2NjI5MDIwOTAsInN1YiI6InN5c3RlbTpzZXJ2aWNlYWNjb3VudDprdWJlcm5ldGVzLWRhc2hib2FyZ
DphZG1pbi11c2VyIn0.HBMZFKOSnfKucamQ_LicDy4YpC9kK6uvAiN7NRUDmuFfKMGth59F5F27c6
HS-L7k-
heTuQBFCra_UQmmg264KUnYiqlfN5BEDRaS_QgBqFNUw6cgR_bjTh_M5wjSFpOHdZ6MLs_MxSlW
xvnnP833RlYAvN0MjeNk2XwzM8IXKMmeaZhhmU9a6nEjJ_6FmoTMLKo-
efoiYNdz4qOSyCmHskocIjVi7L7BLVLMKy_Ij97TaKuW8YtULCDjN_lM0eLmY5t5v61qxexWKm1YF7
NWcOq_dPBdKtyWpVnfEYf2qISLFF5blYt_gFcbvka0Jp0IoSuT0pfjN80qf06T-OoiWw
```

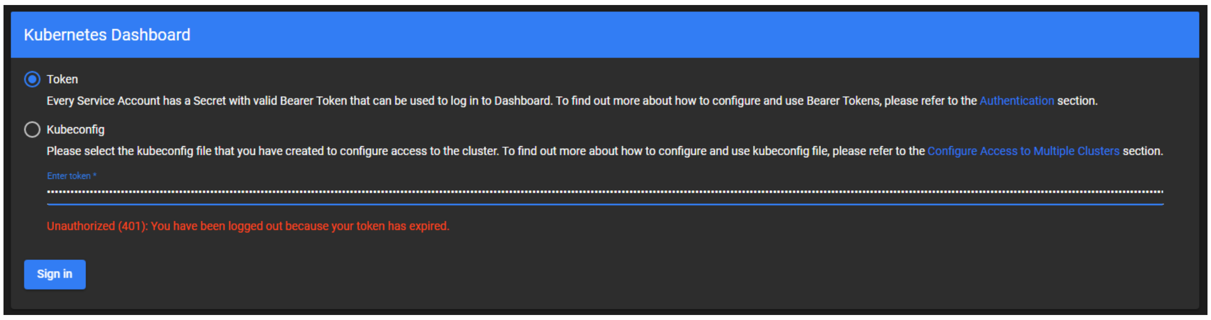
Command line proxy

For accessing the dashboard, first we have to start the dashboard server using the following command:

```
kubectl proxy
```

Kubectl will make Dashboard available at http://localhost:8001/api/v1/namespaces/kubernetes-dashboard/services/https:kubernetes-dashboard:/proxy/.

The UI can *only* be accessed from the machine where the command is executed.


Kubernetes Dashboard

[✓ Token]
Every Service Account has a Secret with valid Bearer Token that can be used to log in to Dashboard. To find out more about how to configure and use Bearer Tokens, please refer to the Authentication section.

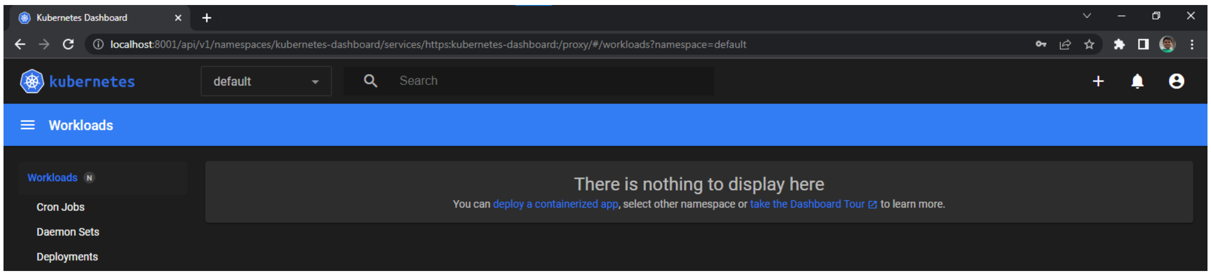
Kubeconfig
Please select the kubeconfig file that you have created to configure access to the cluster. To find out more about how to configure and use kubeconfig file, please refer to the Configure Access to Multiple Clusters section.

Enter token *

Unauthorized (401): You have been logged out because your token has expired.

Welcome view

When we access the Dashboard on an empty cluster, it will display the welcome page. This page contains a button to deploy our first application. In addition to it, we can view which system applications are running by default in the kube-system namespace of our cluster, for example the Dashboard itself.


Kubernetes Dashboard

localhost:8001/api/v1/namespaces/kubernetes-dashboard/services/https:kubernetes-dashboard:/proxy/#/workloads?namespace=default

There is nothing to display here
You can deploy a containerized app, select other namespace or take the Dashboard Tour to learn more.

Q19. Set Up a Kubernetes Cluster, Access application using Kubernetes service.

A Kubernetes service can be used to easily expose an application deployed on a set of pods using a single endpoint.

A service is both a REST object and an abstraction that defines:

1. A set of pods
2. A policy to access them

- Pods in a Kubernetes deployment are regularly created and destroyed, causing their IP addresses to change constantly. This will create discoverability issues for the deployed, application making it difficult for the application frontend to identify which pods to connect.
- This is where the strengths of Kubernetes services come into play: services keep track of the changes in IP addresses and DNS names of the pods and expose them to the end-user as a single IP or DNS.
- Kubernetes services utilize selectors to target a set of pods:

  1. For native Kubernetes applications (which use Kubernetes APIs for service discovery), the endpoint API will be updated whenever there are changes to the pods in the service.

  2. Non-native applications can use virtual-IP-based bridge or load balancer implementation methods offered by Kubernetes to direct traffic to the backend pods.

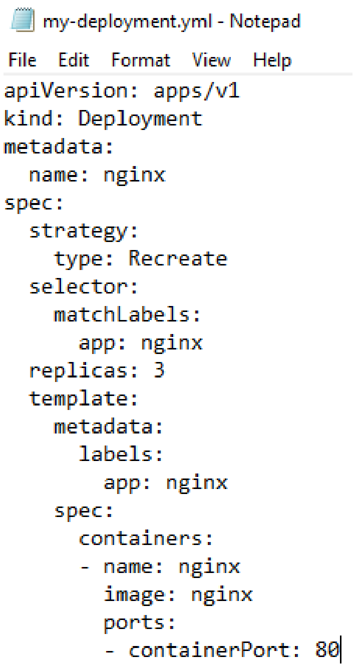
Discovering Kubernetes services

- **DNS:** Here, the DNS server is added to the Kubernetes cluster that watches the Kubernetes API and creates DNS records for each new service.

- **Environment Variables:** In this method, kubelet adds environment variables to Pods for each active service.

Creating services in Kubernetes

- Creating a deployment with the help of a yml configuration file:



```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx
spec:
  strategy:
    type: Recreate
  selector:
    matchLabels:
      app: nginx
  replicas: 3
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx
        ports:
        - containerPort: 80
```

- We then execute the following command to create the deployment:

  kubectl create -f my-deployment.yml

O: deployment.apps/nginx created

- We can get the details of deployment, replicaset and pod using the following commands:

  kubectl get deployment

```
O:  NAME    READY   UP-TO-DATE   AVAILABLE   AGE
    nginx   3/3     3            3           4m48s
```

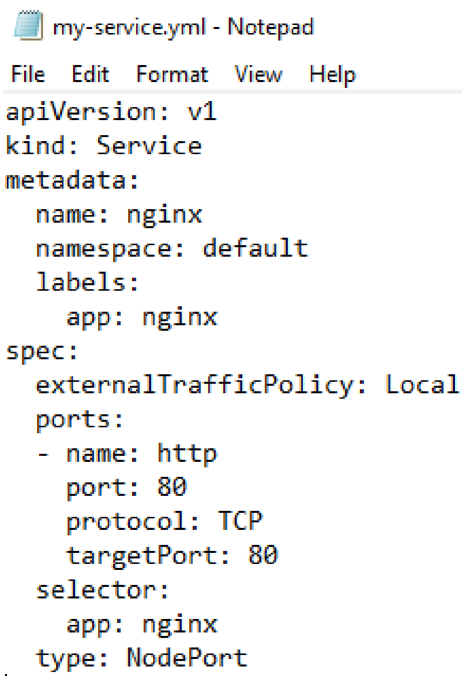
  kubectl get replicaset

```
O:  NAME               DESIRED   CURRENT   READY   AGE
    nginx-6c8b449b8f   3         3         3       5m1s
```

  kubectl get pod

```
O:  NAME                     READY   STATUS    RESTARTS   AGE
    nginx-6c8b449b8f-jz9t7   1/1     Running   0          5m15s
    nginx-6c8b449b8f-nqk7q   1/1     Running   0          5m15s
    nginx-6c8b449b8f-w22d8   1/1     Running   0          5m15s
```

- Next, we will create a Service definition and create it using the following command:



```
apiVersion: v1
kind: Service
metadata:
  name: nginx
  namespace: default
  labels:
    app: nginx
spec:
  externalTrafficPolicy: Local
  ports:
  - name: http
    port: 80
    protocol: TCP
    targetPort: 80
  selector:
    app: nginx
  type: NodePort
```

kubectl create -f my-service.yml

```
O:  service/nginx created
```

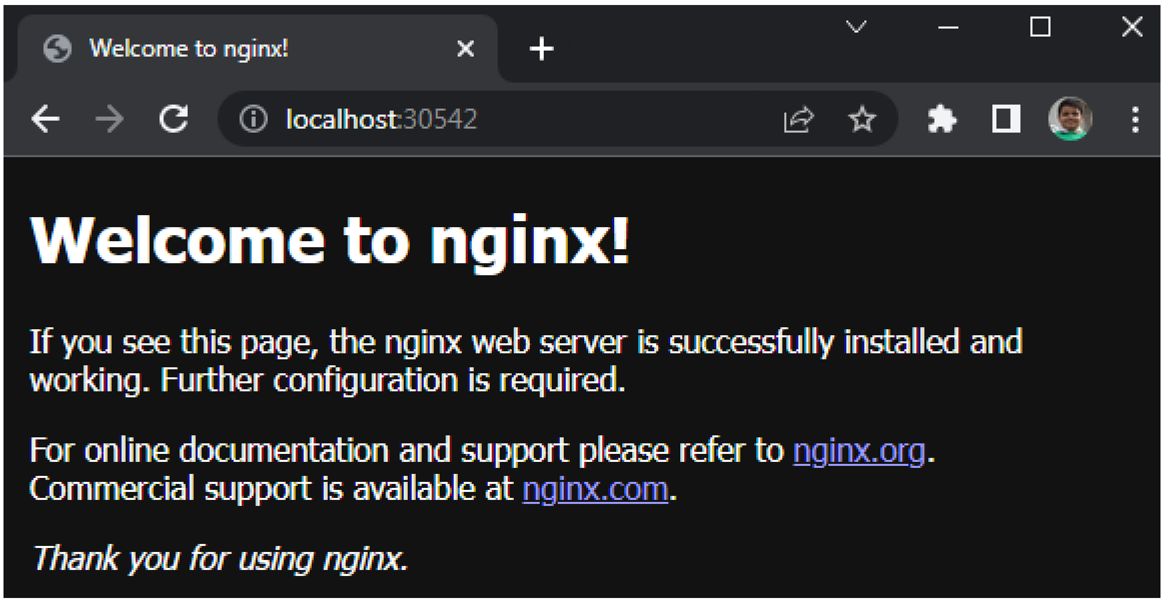
- We use the get service command to obtain to information, especiallyport number on which the service is running:

kubectl get service nginx

```
O:  NAME    TYPE       CLUSTER-IP     EXTERNAL-IP   PORT(S)        AGE

    nginx   NodePort   10.110.47.19   <none>        80:30542/TCP   3m50s
```

- Now, we can see that it is running on port 30542
- The nginx application can be accessed through this service on NodeIp:NodePort
- We are running the service locally, that is why we go to:

localhost:30542



Welcome to nginx!

If you see this page, the nginx web server is successfully installed and working. Further configuration is required.

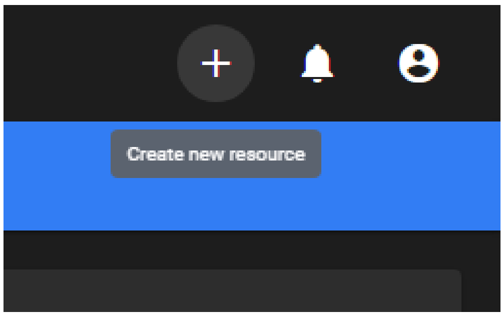
For online documentation and support please refer to nginx.org.
Commercial support is available at nginx.com.

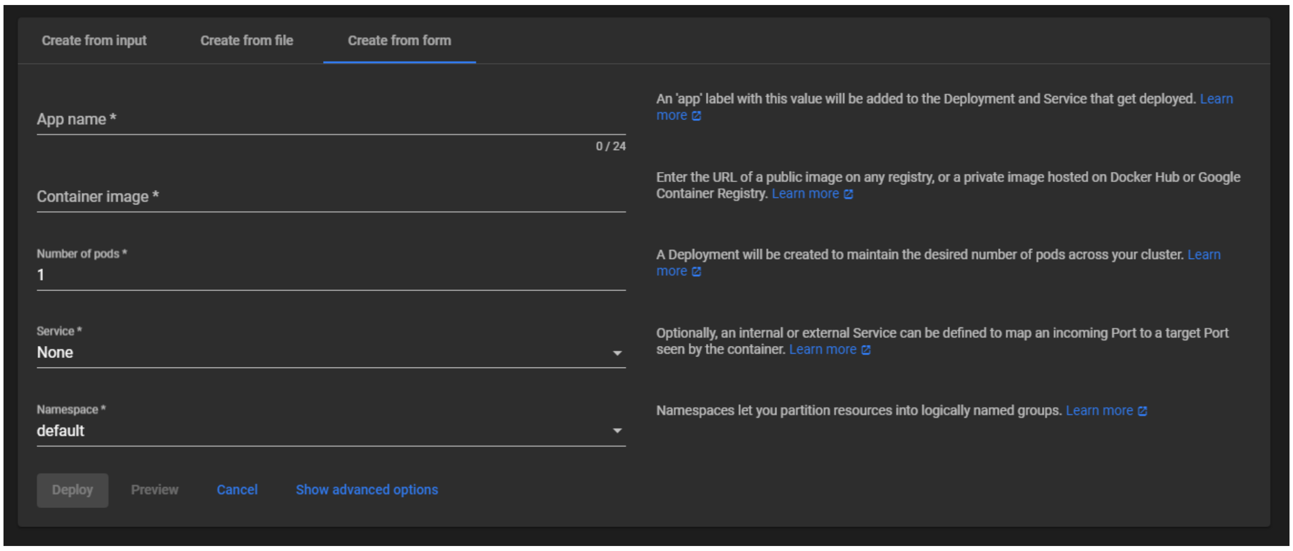
Thank you for using nginx.

Q20. Deploy the website using Dashboard.

Dashboard lets us create and deploy a containerized application as a Deployment and optional Service with a simple wizard. We can either manually specify application details, or upload a YAML or JSON *manifest* file containing application configuration or even write the file directly and execute it.

To open that prompt, we first have to Click the **CREATE** button in the upper right corner of any page to begin.





App name * 0 / 24 — An 'app' label with this value will be added to the Deployment and Service that get deployed. Learn more

Container image * — Enter the URL of a public image on any registry, or a private image hosted on Docker Hub or Google Container Registry. Learn more

Number of pods * 1 — A Deployment will be created to maintain the desired number of pods across your cluster. Learn more

Service * None — Optionally, an internal or external Service can be defined to map an incoming Port to a target Port seen by the container. Learn more

Namespace * default — Namespaces let you partition resources into logically named groups. Learn more

Specifying application details

The deploy wizard expects that you provide the following information:

- **App name** (mandatory): Name for your application. A label with the name will be added to the Deployment and Service, if any, that will be deployed.

The application name must be unique within the selected Kubernetes namespace. It must contain only lowercase letters, numbers and dashes (-). And is limited to 24 characters. Leading and trailing spaces are ignored.

- **Container image** (mandatory): The URL of a public Docker container image on any registry, or a private image (commonly hosted on the Google Container Registry or Docker Hub). The container image specification must end with a colon.

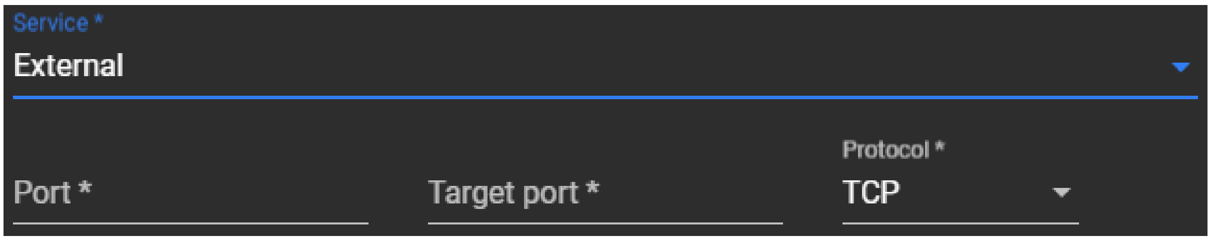
- **Number of pods** (mandatory): The target number of Pods you want your application to be deployed in. The value must be a positive integer.

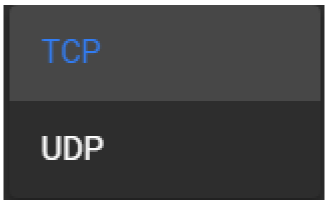
A Deployment will be created to maintain the desired number of Pods across your cluster.

- **Service** (optional): For some parts of your application (e.g. frontends) you may want to expose a Service onto an external, maybe public IP address outside of your cluster (external Service).

Other Services that are only visible from inside the cluster are called internal Services.

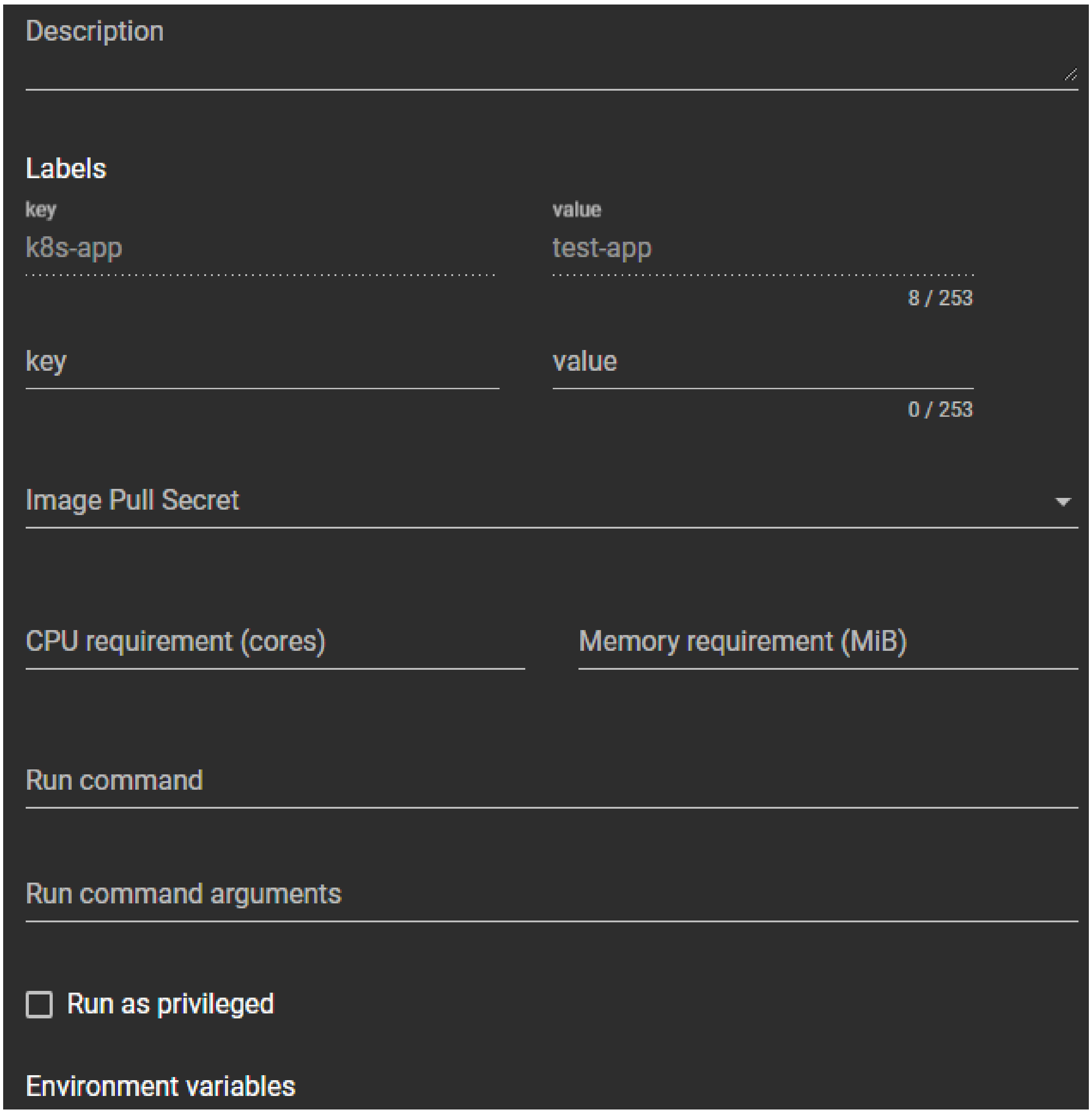
Irrespective of the Service type, if you choose to create a Service and your container listens on a port (incoming), you need to specify two ports. The Service will be created mapping the port (incoming) to the target port seen by the container. This Service will route to your deployed Pods. Supported protocols are TCP and UDP.


Service *
External
Port *
Target port *
Protocol *
TCP



- **Namespace**: Kubernetes supports multiple virtual clusters backed by the same physical cluster. These virtual clusters are called namespaces. They let you partition resources into logically named groups.

If needed, you can expand the **Advanced options** section where you can specify more settings:



Description

Labels

key: k8s-app

value: test-app

8 / 253

key

value

0 / 253

Image Pull Secret

CPU requirement (cores)

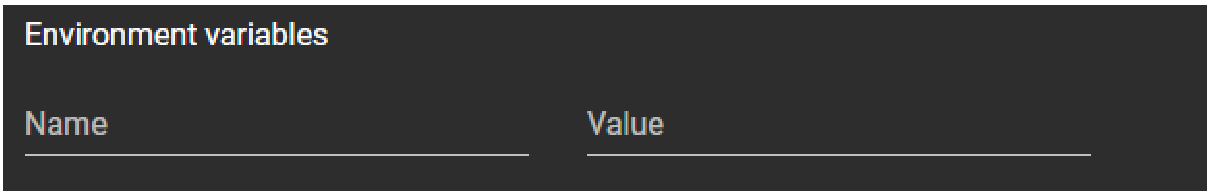
Memory requirement (MiB)

Run command

Run command arguments

☐ Run as privileged

Environment variables



Environment variables

Name

Value

- **Description**: The text you enter here will be added as an annotation to the Deployment and displayed in the application's details.
- **Labels**: Default labels to be used for your application are application name and version. You can specify additional labels to be applied to the Deployment, Service (if any), and Pods, such as release, environment, tier, partition, and release track.

Example:

release=1.0

tier=frontend

environment=pod

track=stable

- **Image Pull Secret**: In case the specified Docker container image is private, it may require pull secret credentials.
- **CPU requirement (cores)** and **Memory requirement (MiB)**: You can specify the minimum resource limits for the container. By default, Pods run with unbounded CPU and memory limits.
- **Run command** and **Run command arguments**: By default, your containers run the specified Docker image's default entrypoint command. You can use the command options and arguments to override the default.
- **Run as privileged**: This setting determines whether processes in privileged containers are equivalent to processes running as root on the host. Privileged containers can make use of capabilities like manipulating the network stack and accessing devices.

Environment variables: Kubernetes exposes Services through environment variables. You can compose environment variable or pass arguments to your commands using the values of environment variables. They can be used in applications to find a Service. Values can reference other variables using the $(VAR_NAME) syntax.