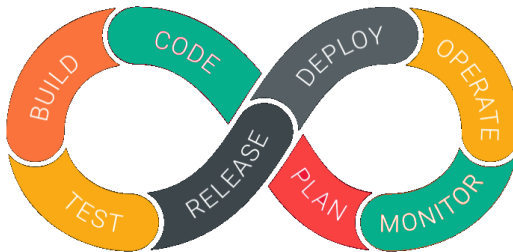


Lab Manual

DevOps Lab

Practical-1	
Aim	Introduction to devops
Course Outcome – 1	Remember the importance of DevOps tools used in software development life cycle
Description	Understanding Devops, Phases, Stages and various online tools
Theory:	<p>DevOps is a culture which promotes collaboration between Development and Operations Team to deploy code to production faster in an automated & repeatable way. The word 'DevOps' is a combination of two words 'development' and 'operations.' DevOps helps to increases an organization's speed to deliver applications and services. It allows organizations to serve their customers better and compete more strongly in the market. In simple words, DevOps can be defined as an alignment of development and IT operations with better communication and collaboration.</p>
Procedure:	<p>DevOps Lifecycle</p>  <p>DevOps is deep integration between development and operations. Understanding DevOps is not possible without knowing DevOps lifecycle. Here is a brief information about the Continuous DevOps life-cycle:</p> <ol style="list-style-type: none"> 1. Development In this DevOps stage the development of software takes place constantly. In this phase, the entire development process is separated into small development cycles. This benefits DevOps team to speed up software development and delivery process. 2. Testing QA team use tools like Selenium to identify and fix bugs in the new piece of code. 3. Integration In this stage, new functionality is integrated with the prevailing code, and testing takes place. Continuous development is only possible due to continuous integration and testing. 4. Deployment In this phase, the deployment process takes place continuously. It is performed in such a manner that any changes made any time in the code, should not affect the functioning of high traffic website. 5. Monitoring In this phase, operation team will take care of the inappropriate system behavior or bugs which are found in production. <p>DevOps Automation Tools</p> <p>It is vital to automate all the testing processes and configure them to achieve speed and agility. This process is known as DevOps automation.</p> <p>The difficulty faced in large DevOps Team that maintain large huge IT infrastructure can be classified briefly into six different categories.</p> <ol style="list-style-type: none"> 1. Infrastructure Automation 2. Configuration Management 3. Deployment Automation

	<ol style="list-style-type: none"> 4. Performance Management 5. Log Management 6. Monitoring. <p>Let's see a few tools in each of these categories and how they solve the pain points—</p> <p>Infrastructure Automation</p> <p>Amazon Web Services (AWS): Being cloud service you do not need to be physically present in the data center. Also, they are easy to scale on-demand. There are no up-front hardware costs. It can be configured to provision more servers based on traffic automatically.</p> <p>Configuration Management</p> <p>Chef: It is a useful DevOps tool for achieving speed, scale, and consistency. It can be used to ease out complex tasks and perform configuration management. With this tool, DevOps team can avoid making changes across ten thousand servers. Instead, they need to make changes in one place which is automatically reflected in other servers.</p> <p>Deployment Automation</p> <p>Jenkins: This tool facilitates continuous integration and testing. It helps to integrate project changes more easily by quickly finding issues as soon as a built is deployed.</p> <p>Log Management</p> <p>Splunk: This is a tool solves the issues like aggregating, storing, and analyzing all logs in one place.</p> <p>Performance Management</p> <p>App Dynamic: It is DevOps tool which offers real-time performance monitoring. The data collected by this tool helps developers to debug when issues occur.</p> <p>Monitoring</p> <p>Nagios: It is also important to make sure people are notified when infrastructure and related services go down. Nagios is one such tool for this purpose which helps DevOps teams to find and correct problems.</p>
Practical-2	
Aim	Version control system using GIT
Course Outcome – 3	Examine the different Version Control strategies
Theory	<p>Version control is a system that records changes to a file or set of files over time so that you can recall specific versions later. For the examples in this book, you will use software source code as the files being version controlled, though in reality you can do this with nearly any type of file on a computer. Git (/git/) is a distributed version-control system for tracking changes in source code during software development. It is designed for coordinating work among programmers, but it can be used to track changes in any set of files.</p>
Procedure:	<p>Git packages are available via apt:</p> <p>From your shell, install Git using apt-get:</p> <pre>2. \$ sudo apt-get update \$ sudo apt-get install git</pre> <p>Verify the installation was successful by typing git --version:</p> <pre>4. \$ git --version git version 2.9.2</pre>

Configure your Git username and email using the following commands, replacing Emma's name with your own. These details will be associated with any commits that you create:

```
6. $ git config --global user.name "Emma Paris"
$ git config --global user.email "eparis@atlassian.com"
```

Fedora (dnf/yum)

Git packages are available via both [yum](#) and [dnf](#):

From your shell, install Git using dnf (or yum, on older versions of Fedora):

```
$ sudo dnf install git
```

or

```
$ sudo yum install git
```

Verify the installation was successful by typing git --version:

```
3. $ git --version
git version 2.9.2
```

Configure your Git username and email using the following commands, replacing Emma's name with your own. These details will be associated with any commits that you create

```
5. $ git config --global user.name "Emma Paris"
$ git config --global user.email "eparis@atlassian.com"
```

Build Git from source on Linux

Debian / Ubuntu

Git requires the several dependencies to build on Linux. These are available via [apt](#):

From your shell, install the necessary dependencies using apt-get:

```
2. $ sudo apt-get update
$ sudo apt-get install libcurl4-gnutls-dev libexpat1-dev gettext libz-dev libssl-dev asciidoc xmlto
docbook2x
```

Clone the Git source (or if you don't yet have a version of Git installed, [download and extract it](#)):

```
$ git clone https://git.kernel.org/pub/scm/git/git.git
```

To build Git and install it under /usr, run make:

```
5. $ make all doc info prefix=/usr
$ sudo make install install-doc install-html install-info install-man prefix=/usr
```

Fedora

Git requires the several dependencies to build on Linux. These are available via both [yum](#) and [dnf](#):

From your shell, install the necessary build dependencies using dnf (or yum, on older versions of Fedora):

```
$ sudo dnf install curl-devel expat-devel gettext-devel openssl-devel perl-devel zlib-devel
asciidoc xmlto docbook2X
```

or using yum. For yum, you may need to install the Extra Packages for Enterprise Linux (EPEL) repository first:

	<pre>\$ sudo yum install epel-release \$ sudo yum install curl-devel expat-devel gettext-devel openssl-devel perl-devel zlib-devel asciidoc xmlto docbook2X</pre> <p>Symmlink docbook2X to the filename that the Git build expects:</p> <pre>\$ sudo ln -s /usr/bin/db2x_docbook2texi /usr/bin/docbook2x-texi</pre> <p>Clone the Git source (or if you don't yet have a version of Git installed, download and extract it):</p> <pre>\$ git clone https://git.kernel.org/pub/scm/git/git.git</pre> <p>To build Git and install it under /usr, run make:</p> <pre>5. \$ make all doc prefix=/usr \$ sudo make install install-doc ins</pre>
Practical-3	
Aim	Cloning and Branching using Git
Course Outcome – 3	Examine the different Version Control strategies
Theory	<p>git clone is primarily used to point to an existing repo and make a clone or copy of that repo at in a new directory, at another location. The original repository can be located on the local filesystem or on remote machine accessible supported protocols. The git clone command copies an existing Git repository. Team members publish, share, review, and iterate on code changes through Git branches shared with others. Adopt a branching strategy for your team. You can collaborate better and spend less time managing version control and more time developing code.</p>
Procedure	<p>You can use Sourcetree, Git from the command line, or any client you like to clone your Git repository. These instructions show you how to clone your repository using Git from the terminal.</p> <p>From the repository, click + in the global sidebar and select Clone this repository under Get to work.</p> <p>Copy the clone command (either the SSH format or the HTTPS).</p> <p>If you are using the SSH protocol, ensure your public key is in Bitbucket and loaded on the local system to which you are cloning.</p> <p>From a terminal window, change to the local directory where you want to clone your repository. Paste the command you copied from Bitbucket, for example:</p> <p>CLONE OVER HTTPS:</p> <pre>\$ git clone https://username@bitbucket.org/teamsinspace/documentation-tests.git</pre> <p>CLONE OVER SSH:</p> <pre>\$ git clone ssh://git@bitbucket.org:teamsinspace/documentation-tests.git</pre> <p>If the clone was successful, a new sub-directory appears on your local drive in the directory where you cloned your repository. This directory has the same name as the Bitbucket repository that you cloned. The clone contains the files and metadata that Git requires to maintain the changes you make to the source files.</p> <p>From the repository, click + in the global sidebar and select Create a branch under Get to work.</p>

From the popup that appears, select a **Type** (if using the [Branching model](#)), enter a **Branch name** and click **Create**.

Note

Whenever you create a branch from Bitbucket or from an issue in Jira Software, Bitbucket removes characters that are invalid in references, file systems or shell, and replace them with a valid substitute.

After you create a branch, you need to check it out from your local system. Use the fetch and checkout commands that Bitbucket provides, similar to the following:

```
$ git fetch && git checkout <feature>
```

Make your changes locally and then add, commit, and push your changes to the <feature> branch:

```
$ git add .
```

```
$ git commit -m "adding a change from the feature branch"
```

```
$ git push origin <feature>
```

Click the **Source** page of your repository. You should see both the master and the <feature> branch in the branches dropdown. When you make commits to the feature branch, you'll see the files specific to that branch.

To create a branch locally

You can create a branch locally as long as you have a cloned version of the repo. From your terminal window, list the branches on your repository.

```
$ git branch
```

```
* master
```

This output indicates there is a single branch, the master and the asterisk indicates it is currently active.

Create a new feature branch in the repository

```
$ git branch <feature_branch>
```

Switch to the feature branch to work on it.

```
$ git checkout <feature_branch>
```

You can list the branches again with the git branch command.

Commit the change to the feature branch:

```
$ git add .
```

```
$ git commit -m "adding a change from the feature branch"
```

Switch back to the master branch.

```
$ git checkout master
```

Push the feature branch to Bitbucket:

```
$ git push origin <feature_branch>
```

	View the Source page of your repository in Bitbucket. You should see both the master and the feature branch. When you select the feature branch, you see the Source page from that perspective. Select the feature branch to view its Recent commits .
Practical-4	
Aim	Installation and configuration of Docker
Course Outcome – 4	Analyze & Illustrate the Containerization of OS images and deployment of applications over Docker
Theory	Docker is a tool designed to make it easier to create, deploy, and run applications by using containers. Containers allow a developer to package up an application with all of the parts it needs, such as libraries and other dependencies, and deploy it as one package. By doing so, thanks to the container, the developer can rest assured that the application will run on any other Linux machine regardless of any customized settings that machine might have that could differ from the machine used for writing and testing the code.
Procedure	<ol style="list-style-type: none"> Update the apt package index and install packages to allow apt to use a repository over HTTPS: <pre> 2. \$ sudo apt-get update 3. 4. \$ sudo apt-get install \ 5. apt-transport-https \ 6. ca-certificates \ 7. curl \ 8. gnupg-agent \ 9. software-properties-common </pre> <ol style="list-style-type: none"> Add Docker's official GPG key: <pre> 11. \$ curl -fsSL https://download.docker.com/linux/ubuntu/gpg sudo apt-key add - </pre> <p>Verify that you now have the key with the fingerprint 9DC8 5822 9FC7 DD38 854A E2D8 8D81 803C 0EBF CD88, by searching for the last 8 characters of the fingerprint.</p> <pre> \$ sudo apt-key fingerprint 0EBFCD88 pub rsa4096 2017-02-22 [SCEA] 9DC8 5822 9FC7 DD38 854A E2D8 8D81 803C 0EBF CD88 uid [unknown] Docker Release (CE deb) <docker@docker.com> sub rsa4096 2017-02-22 [S] </pre> <ol style="list-style-type: none"> Use the following command to set up the stable repository. To add the nightly or test repository, add the word nightly or test (or both) after the word stable in the commands below. Learn about nightly and test channels. Note: The <code>lsb_release -cs</code> sub-command below returns the name of your Ubuntu distribution, such as xenial. Sometimes, in a distribution like Linux Mint, you might need to change <code>\$(lsb_release -cs)</code> to your parent Ubuntu distribution. For example, if you are using Linux Mint Tessa, you could use bionic. Docker does not offer any guarantees on untested and unsupported Ubuntu distributions. <ul style="list-style-type: none"> x86_64 / amd64

- armhf
- arm64
- ppc64le (IBM Power)
- s390x (IBM Z)

```
$ sudo add-apt-repository \
"deb [arch=amd64] https://download.docker.com/linux/ubuntu \
$(lsb_release -cs) \
stable"
```

INSTALL DOCKER ENGINE

1. Update the apt package index, and install the *latest version* of Docker Engine and containerd, or go to the next step to install a specific version:

2. `$ sudo apt-get update`
3. `$ sudo apt-get install docker-ce docker-ce-cli containerd.io`

Got multiple Docker repositories?

If you have multiple Docker repositories enabled, installing or updating without specifying a version in the apt-get install or apt-get update command always installs the highest possible version, which may not be appropriate for your stability needs.

4. To install a *specific version* of Docker Engine, list the available versions in the repo, then select and install:

- a. List the versions available in your repo:

```
$ apt-cache madison docker-ce
```

```
docker-ce | 5:18.09.1~3-0~ubuntu-xenial | https://download.docker.com/linux/ubuntu
xenial/stable amd64 Packages
docker-ce | 5:18.09.0~3-0~ubuntu-xenial | https://download.docker.com/linux/ubuntu
xenial/stable amd64 Packages
docker-ce | 18.06.1~ce~3-0~ubuntu      | https://download.docker.com/linux/ubuntu
xenial/stable amd64 Packages
docker-ce | 18.06.0~ce~3-0~ubuntu      | https://download.docker.com/linux/ubuntu
xenial/stable amd64 Packages
...
```

- b. Install a specific version using the version string from the second column, for example, 5:18.09.1~3-0~ubuntu-xenial.

```
$ sudo apt-get install docker-ce=<VERSION_STRING>
docker-ce-cli=<VERSION_STRING> containerd.io
```

5. Verify that Docker Engine is installed correctly by running the hello-world image.

6. `$ sudo docker run hello-world`

This command downloads a test image and runs it in a container. When the container runs, it prints an informational message and exits.

Docker Engine is installed and running. The docker group is created but no users are added to it. You need to use sudo to run Docker commands. Continue to [Linux postinstall](#) to allow non-privileged users to run Docker commands and for other optional configuration steps.

	<p><i>UPGRADE DOCKER ENGINE</i></p> <p>To upgrade Docker Engine, first run <code>sudo apt-get update</code>, then follow the installation instructions, choosing the new version you want to install.</p> <p>Install from a package</p> <p>If you cannot use Docker's repository to install Docker Engine, you can download the .deb file for your release and install it manually. You need to download a new file each time you want to upgrade Docker.</p> <ol style="list-style-type: none"> 1. Go to https://download.docker.com/linux/ubuntu/dists/, choose your Ubuntu version, then browse to pool/stable/, choose amd64, armhf, arm64, ppc64el, or s390x, and download the .deb file for the Docker Engine version you want to install. Note: To install a nightly or test (pre-release) package, change the word stable in the above URL to nightly or test. Learn about nightly and test channels. 2. Install Docker Engine, changing the path below to the path where you downloaded the Docker package. <div> 3. <code>\$ sudo dpkg -i /path/to/package.deb</code> </div> <p>The Docker daemon starts automatically.</p> <ol style="list-style-type: none"> 4. Verify that Docker Engine is installed correctly by running the hello-world image. <div> 5. <code>\$ sudo docker run hello-world</code> </div> <p>This command downloads a test image and runs it in a container. When the container runs, it prints an informational message and exits.</p> <p>Docker Engine is installed and running. The docker group is created but no users are added to it. You need to use sudo to run Docker commands. Continue to Post-installation steps for Linux to allow non-privileged users to run Docker commands and for other optional configuration steps.</p> <p><i>UPGRADE DOCKER ENGINE</i></p> <p>To upgrade Docker Engine, download the newer package file and repeat the installation procedure, pointing to the new file.</p>
Practical-5	
Aim	Creating Dockerfile and Docker Volume
Course Outcome - 4	Analyze & Illustrate the Containerization of OS images and deployment of applications over Docker
Theory	<p>Volumes in Docker is a mechanism which enables us to generate data into host machine directory. Volumes are managed mainly by docker daemon, on the other hand, bind mounts is created on the user managed directory on the host machine.</p> <p>Docker can build images automatically by reading the instructions from a Dockerfile. A Dockerfile is a text document that contains all the commands a user could call on the command line to assemble an image. Using docker build users can create an automated build that executes several command-line instructions in succession.</p> <p>This page describes the commands you can use in a Dockerfile. When you are done reading this page, refer to the Dockerfile Best Practices for a tip-oriented guide.</p>
procedure	Please browse through the file, it's enough for now if you note the key structural elements, skim through the comments and see how the file is built up on a high level.

```
<script  
src="https://gist-it.appspot.com/https://github.com/docker-library/php/blob/f4baf0edbc4e05e241938c68bcc7c9635707583d/7.2/stretch/apache/Dockerfile"></script>
```

At this point I would like you to understand the following key points based on the example file:

1. The Dockerfile is a text file that (mostly) contains the instructions that you would execute on the command line to create an image.
 2. A Dockerfile is a step by step set of instructions.
 3. Docker provides a set of standard instructions to be used in the Dockerfile, like FROM, COPY, RUN, ENV, EXPOSE, CMD just to name a few basic ones.
 4. Docker will build a Docker image automatically by reading these instructions from the Dockerfile.
- FROM
 - COPY vs ADD
 - ENV
 - RUN
 - VOLUME
 - USER
 - WORKDIR
 - EXPOSE
 - CMD and ENTRYPOINT
 - ONBUILD

Volumes are the preferred way to persist data in Docker containers and services. Some use cases for volumes include:

- Sharing data among multiple running containers. If you don't explicitly create it, a volume is created the first time it is mounted into a container. When that container stops or is removed, the volume still exists. Multiple containers can mount the same volume simultaneously, either read-write or read-only. Volumes are only removed when you explicitly remove them.
- When the Docker host is not guaranteed to have a given directory or file structure. Volumes help you decouple the configuration of the Docker host from the container runtime.
- When you want to store your container's data on a remote host or a cloud provider, rather than locally.
- When you need to back up, restore, or migrate data from one Docker host to another, volumes are a better choice. You can stop containers using the volume, then back up the volume's directory (such as `/var/lib/docker/volumes/<volume-name>`).

Good use cases for bind mounts

In general, you should use volumes where possible. Bind mounts are appropriate for the following types of use case:

- Sharing configuration files from the host machine to containers. This is how Docker provides DNS resolution to containers by default, by mounting `/etc/resolv.conf` from the host machine into each container.
- Sharing source code or build artifacts between a development environment on the Docker host and a container. For instance, you may mount a Maven `target/` directory into a container, and each time you build the Maven project on the Docker host, the container gets access to the rebuilt artifacts.

If you use Docker for development this way, your production Dockerfile would copy the production-ready artifacts directly into the image, rather than relying on a bind mount.

- When the file or directory structure of the Docker host is guaranteed to be consistent with the bind mounts the containers require.

Good use cases for tmpfs mounts

tmpfs mounts are best used for cases when you do not want the data to persist either on the host machine or within the container. This may be for security reasons or to protect the performance of the container when your application needs to write a large volume of non-persistent state data.

Tips for using bind mounts or volumes

If you use either bind mounts or volumes, keep the following in mind:

- If you mount an **empty volume** into a directory in the container in which files or directories exist, these files or directories are propagated (copied) into the volume. Similarly, if you start a container and specify a volume which does not already exist, an empty volume is created for you. This is a good way to pre-populate data that another container needs.
- If you mount a **bind mount or non-empty volume** into a directory in the container in which some files or directories exist, these files or directories are obscured by the mount, just as if you saved files into /mnt on a Linux host and then mounted a USB drive into /mnt. The contents of /mnt would be obscured by the contents of the USB drive until the USB drive were unmounted. The obscured files are not removed or altered, but are not accessible while the bind mount or volume is mounted.