**DevOps**

**Git Command Line operations**

-**git config --global user.name "KweeTeck** " ( user enters their name)

-**git config --global user.email tan\_kwee\_teck@ichat.sp.edu.sg** (user enter their email address)

-**git config –list** (to verify and check that Git has saved the new user settings)

-**cd** **"C:\Local\_Git\_Repository"** (change to the directory where you want to create a new Git repository)

-**mkdir** **lab1** (add a new file in ur folder)

-**git init** (initialize a new Git repository and create a .git in the file. It's important to note that "git init" only needs to be **run once** per repository. If you've already initialized a repository in a directory, running "git init" again will **overwrite** your existing repository with a new, empty one. Take note that **“.git”** is a **“hidden item”,** therefore please make sure that you configure your File Explorer to **show “hidden items”.**

-**git status** (to check the **status** of the Git repository)

-**git add** **HelloWorld.py** (Before **committing a new or modified file** to the Git repository, we need to **add** this file to the “**staging area**”)

-**git commit -m** **"random message"** (**“commit”** this file to the Git repository with a short text **comment** that **describes the changes** made in this commit)

-**git diff** (to view the file **differences** before deciding to either commit the file again to the Git repository or continue modifying the file and commit the changes to the Git repository later.)

-**git branch** (To **view** all the **branches** **created** in the current Git repository)

-**git branch** **"bug-fix1"**(**create** **new branches** in Git repository. Use the git branch command with a **parameter** specifying the **name** of the **new branch**)

-**git checkout** **"bug-fix1"** (**switch** to the **new branch**)

-**git merge** **"bug-fix1"(**To **merge** a **branch** into the **current branch**. Need to **switch back** to “**master**” branch **before** **merge**)

--**git remote –v** (To **check** if the **current local Git repository** has the remote repository server **correctly**

**configured**)

-**git remote add origin** [**https://GitHub.com/kweetecktan-ichat/Lab1.git**](https://GitHub.com/kweetecktan-ichat/Lab1.git)(**add** a **remote repository** to your **local Git repository**. The word "**origin**" is a commonly used name for the remote repository, but you can **choose a different name** if desired)

-**git remote set-url origin {URL}** (used to **change** the **URL** of a **remote repository** in Git. It **updates** the **URL** of the remote repository named "origin" to the specified URL.)

-**git push --set-upstream origin master** (This command is used to **push** your **local branch** named **master** to the **remote repository** and set it as the **upstream branch**. It explicitly specifies the branch name master as the local branch to push. You can **replace master** with the **name of your local branch**.

**After** running this command, you can use **git push** in future pushes **without specifying the remote** and **branch names**. Git will use the **configured upstream branch** as the **default destination**)

-**git push** (This command is used to **push** your local branch to the **remote** **repository**. It **assumes** that you have **already set up the upstream branch** for your local branch using **git push -u origin** or **git push –set upstream**. If you **haven't configured** the upstream branch, Git will display an **error message** and suggest using git push --set-upstream to set it)

-**git push –u origin** (This command is used to **push** your **local branch** to the **remote repository** and set it as the **upstream branch**. It **assumes** that you are **currently** on the **branch** you want to **push**. **After** running this command, you can use **git push** in future pushes **without specifying** the remote and branch names. Git will use the **configured upstream branch** as the **default** destination)

-**git tag –a v1.0 –m "Initial release v1.0”**( is used to **create** an annotated **tag** in Git with the **name** **"v1.0"** and a **message** **"Initial release v1.0"** describing the tag)

-**git push origin v1.0** (is used to **push** a **specific tag**)

-**git submodule add** **<Git submodule repository URL>** (is used to **add** a **new submodule** to your Git repository)

-**git clone** **<URL of repository to clone>** (To **clone** a **Git repository**. This command will **create** a **local** **copy** of the repository in a directory with the same name as the repository. The repository's files, commit history, and branches will be copied to your local machine.)

-**git log** (to **view** the **commit history** of a Git repository)

-**git clone --recurse-submodule**(to clone a Git repository and its submodules recursively.)

**-GitHub** (**cloud-based remote Git repository** where developers can **upload** their **software** projects for collaboration and **sharing** with other developers)

**Docker**

Docker Workflow

1. Sends “Docker container run” command from CLI

2. Received command at API endpoint. Instructs container to create new container.

3. Sends to run the image. Instruct run to create a container.

4. Creates a container.

Docker Commands

1. docker image ls (list the images)
2. docker image pull ubuntu (pull an image)

docker image pull <registry>/<respository>:<tag>

1. docker image build -t mywebapp . ("." indicates that the Dockerfile and the build context are in the current directory.)
2. docker image rm redis (remove image, only can remove the image when there is no container running on it]
3. docker container run -d –name web –p 80:8080 mywebapp ("-p 80:8080" is an option used to publish a container's port to the host. In this case, it maps port 80 on the host to port 8080 in the container.)
4. docker container ls (list currently running container)
5. docker container ls -a (list all container in docker including running and stopping containers)
6. docker container run -it --name my-container alpine /bin/sh (assign a name to the container and run the container)
7. docker container exec -it (container name /ID) /bin/sh (execute command within the container/getting inside the running container)
8. docker container stop (container name /ID) (stop the container)
9. docker container start (container name /ID) (start the container)
10. Ctrl+P+Q (to exit from the container without stopping it)
11. Exit (exit the container and terminal it)
12. Docker container inspect (container name/ID)(inspect the container)
13. docker container run -d --name my-container alpine top

(d=detach means container will run at the background)

(top is the command will be execute within the container)

1. docker container attach my-container (attach to run the container, Ctrl+C to exit)
2. docker container run -it ubuntu /bin/bash (run/login to the container)

[ i-keep STDIN open even if not attached, t- allocate to pseudo-tty (terminal)]

(make the container interactive and attach it to your terminal)

[/bin/bash-main processor to run the container/shell]

1. ps aux (to list out running processor
2. )
3. docker container ls -aq (return all the container ID)
4. cat clean.sh (short form for docker stop $(docker container ls -aq), docker rm $(docker container ls -aq), docker rmi $(docker images -q) )[to stop, remove all containers and images]
5. docker login (key in ur username and password)
6. docker image tag nginx skkt/docker101 (create a tag with the name of the repository is docker101)
7. docker image push skkt/docker101 (push the tag to your repository)
8. docker image tag nginx skkt/docker101:version2 (create another tag with name of version2)
9. docker image push skkt/docker101:version2 (push the version2 tag to the repository)
10. echo “I love docker” > message (create a message inside the container)
11. ls -l (it is used to list the files and directories in a long format, providing detailed information about each entry.)
12. cat message (to read/display the message)
13. mkdir -p (directory name) (create a directory in the container host filesystem)
14. touch (update the access and modification timestamps of a file or create an empty file if it doesn't exist.)
15. ls -lrt (to list files and directories in a directory in a long format, sorted by their modification time in ascending order)
16. docker container run -it --name my-container -v $HOME/my-bind-mouth:/tmp alpine /bin/sh
17. docker volume create myvol (create a volume)
18. docker volume ls (list currently running volume)
19. docker container run -it –name vol-container\--mount type=volume,source=myvol,target=/tmp\alpine /bin/sh (run a container and mount the volume)
20. docker volume inspect myvol (inspect the volume)
21. sudo ls -l (mountpoint address) (list the mountpoint)
22. ls -l /tmp (list the files in the volume mounted directory)
23. docker volume rm myvol (remove volume)
24. docker-compose up-d (create and start the container)
25. docker compose ps (list all the containers in the compose)
26. docker compose images (list all the images)
27. docker compose stop (stop all the containers in the compose)
28. docker-compose up -d –sclae db=3 (scale up the db=database)
29. go (to check if go is install)
30. apk add go (to add go)
31. docker commit myapp alpine-go (create new image template for the container)
32. vim Dockerfile create a simple dockerfile, press esc and :wq! to save and quit)
33. docker save hellobackup > hellobackup.tar (save the images)
34. docker load < hellobackup.tar (to load the image)

Kubernetes’s commands

1. ls (list all the pods)
2. kubectl create -f nginx-pod.yaml (create a new pod)
3. kubectl get pods (get the list of pod that currently run on the namespace)
4. kubectl describe pod webapp-pod (get the details of the pod)
5. kubectl delete pod webapp-pod (delete the pod)
6. kubectl create -f replicaset-1.yaml (create replicaset command)
7. kubectl get replicaset (get all replicaset)
8. kubectl describe replicaset replicaset-1 (describe replicaset command and get the details)
9. kubectl apply -f replicaset-1.yaml (apply the changes to the yaml file)
10. kubectl scale replicaset replicaset-1 –replicas=5 (scale the number of replicas)
11. kubectl apply -f deployment-1 (edit deployment command /rolling update)
12. kubectl set image deployment payment-deployment nginx=nginx:1.19.1 (update the image of container)
13. kubectl rollout status deployment payment-deployment (show us the message whether the deployment has been rolled up successful)
14. kubectl rollout undo deployment payment-deployment (destroy the error)
15. kubectl rollout history deployment payment-deployment (show the history of the deployment records)
16. kubectl set image deployment payment-deployment nginx=nginx:1.19.1 –record (input the change cost)
17. kubectl rollout history deployment payment-deployment --revision=5 (deeper looking at the record)
18. kubectl rollout pause deployment payment-deployment ( pause the rollout of the deployment)
19. kubectl get service my-service (get service details)
20. kubectl get endpoint (retrieve information about the endpoints in your Kubernetes cluster)
21. kubectl get pods -o wide (allows you to retrieve information about pods in your Kubernetes cluster, including additional details such as node and IP information.)
22. curl IP address (get detail of our container)
23. kubectl expose pod service-pod-2 –name=my-service-2 –port=80 (to create a service for an existing pod in a Kubernetes cluster)