**GIT Command**

|  |  |
| --- | --- |
| Command | Use |
| git init | This command turns a directory into an empty Git repository. |
| git add ./filename -all | Add file/s into the staging area |
| git commit -m | Save changes from staging area to local repository |
| git status | show the current status of files red color : Untracked files, Green color : added files into staging area |
| git config | git config --global user.email "my@emailaddress.com" |
|  | git config --global user.name "Brian Kerr" |
| git log | show the committed changes history |
| git diff | Compare changes of working directory with Staging area(b stands for staging area) |
| git diff --staged | compare changes of Staging area with Local repository |
| git diff --HEAD | Compare changes of working directory with Local repository |
| git restore <File Name> | Revert changes from working directory option #1 |
| git checkout --<File Name> | Revert changes from working directory Option #2 |
| git restore --staged <file name>  git restore <File name> | revert from staging to working area  restore from working area |
| git reset HEAD~  git restore <File name> | revert changes from local repo to working directory |
| git branch | To determine what branch the local repository is on, add a new branch, or delete a branch |
| git branch <branch\_name> | Create a new branch |
| git branch -a | List all remote or local branches |
| git branch -d <branch\_name> | Delete a branch |
| git checkout | To start working in a different branch, use *git checkout* to switch branches. |
| git merge | Integrate branches together. *git merge* combines the changes from one branch to another branch |
| git remote | To connect a local repository with a remote repository |
|  | git remote add origin git@account\_name.git.beanstalkapp.com:/acccount\_name/repository\_name.git |
| git remote -v | List named remote repositories |
| git clone | To create a local working copy of an existing remote repository |
|  | git clone [git@account\_name.git.beanstalkapp.com:/acccount\_name/repository\_name.git](mailto:git@account_name.git.beanstalkapp.com:/acccount_name/repository_name.git) |
| git pull | This pulls the changes from the remote repository to the local computer. |
|  | git pull origin staging |
| git push | Sends local commits to the remote repository. *git push* requires two parameters: the remote repository and the branch that the push is for. |
|  | git push origin staging |
| git stash | To save changes made when they’re not in a state to commit them to a repository. |
|  | Store current work with untracked files git stash -u |
|  | Bring stashed work back to the working directory git stash pop |
| git reflog | Git keeps track of updates to the tip of branches using a mechanism called reflog. This allows you to go back to changesets even though they are not referenced by any branch or tag. |
|  |  |

DOCKER Command

Docker Terms

**Microservices** us that same type of applications become easier to build and maintain when they are broken down into smaller,composable pieces which work together. Each components is developed separately, and the application is then simply the sum of its constituent components.

For example an online shop with separate microservices for user account, product catalog order processing and shopping carts.

**Hypervisor:** Hypervisor is a piece of software or firmware that creates and run virtual machine. A hypervisor is sometimes also called virtual machine manager<VMM)

**Docker**

* Docker is a tool to make it easier to create,deploy and run applications by using containers.
* Docker containers are lightweight alternatives to virtual Machines and its uses the host OS
* You dont have to pre-allocate any RAM in containers
* Docker is Advanced Version of Virtualization

**Docker Edition:** Community and Enterprise

**Docker Image**

* Read only Template used to create containers.
* Contains all application Dependencies
* Build by Docker Users
* Stored In Docker hub on your Local Registry

**Docker Container**

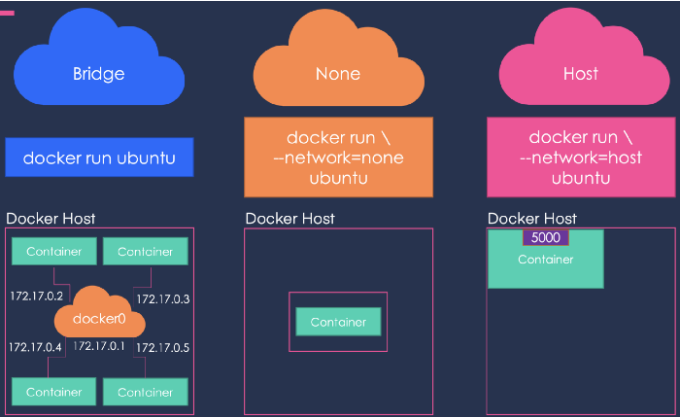
* Isolated Application Platform
* Contains Everything Need to Run The Application.
* Share same OS
* Docker Network

Docker networking **allows you to attach a container to as many networks as you like**

**1] bridge**: The default network driver. If you don’t specify a driver, this is the type of network you are creating. Bridge networks are usually used when your applications run in standalone containers that need to communicate.

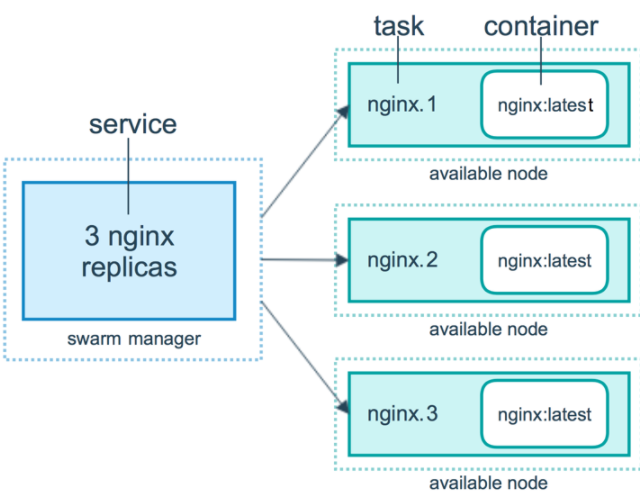
**2] host:** For standalone containers, remove network isolation between the container and the Docker host, and use the host’s networking directly.

**3] overlay:** Overlay networks connect multiple Docker daemons together and enable swarm services to communicate with each other. You can also use overlay networks to facilitate communication between a swarm service and a standalone container, or between two standalone containers on different Docker daemons.



**Docker Swarm**

Docker swarm is a container orchestration tool, meaning that it allows the user to manage multiple containers deployed across multiple host machines. One of the key benefits associated with the operation of a docker swarm is the high level of availability offered for applications.



**Docker namespace and cgroups**

**Namespace:** wraps a global system resource in an abstraction that makes it appear to the processes within the namespace that they have their own isolated instance of the global resource. Pid, net,mnt,uts

**cgroup**: Control Groups provide a mechanism for aggregating/partitioning sets of tasks, and all their future children, into hierarchical groups with specialized behaviour.

Memory, CPU,Network

**Docker-Machine** is a tool that lets you install Docker Engine on Virtual Hosts.

Docker Command

|  |  |
| --- | --- |
| docker –version |  |
| docker pull <image name> | pull images from the docker repository(hub.docker.com) |
| docker run -it -d <image name> | This command is used to create a container from an image |
| docker ps | This command is used to list the running containers |
| docker ps -a | This command is used to show all the running and exited containers |
| docker exec -it <container id> bash | This command is used to access the running container |
| docker stop <container id> | This command stops a running container |
| docker kill <container id> | This command kills the container by stopping its execution immediately |
| docker commit <conatainer id> <username/imagename> | This command creates a new image of an edited container on the local system |
| docker login | This command is used to login to the docker hub repository |
| docker push <username/image name> | This command is used to push an image to the docker hub repository |
| docker images | This command lists all the locally stored docker images |
| docker rm <container id> | This command is used to delete a stopped container |
| docker rmi <image-id> | This command is used to delete an image from local storage |
| docker build <path to docker file> | This command is used to build an image from a specified docker file` |

**Network**

|  |  |
| --- | --- |
| docker network ls | lists the details of all the network in the cluster. |
| connect | Connect a container to a network |
| create | Create a network |
| disconnect | Disconnect a container from a network |
| inspect | Display detailed information on one or more networks |
| ls | List networks |
| prune | Remove all unused networks |
| rm | Remove one or more networks |
|  |  |

**Dockerfile**

|  |  |
| --- | --- |
| **FROM** | specifies the base(parent) image. |
| RUN | runs a Linux command. Used to install packages into container, create folders, etc |
| ENV | sets environment variable. We can have multiple variables in a single dockerfile. |
| EXPOSE | expose ports |
| ENTRYPOINT | provides command and arguments for an executing container. |
| CMD | provides a command and arguments for an executing container. There can be only one CMD. |
| VOLUME | create a directory mount point to access and store persistent data. |
| WORKDIR | sets the working directory for the instructions that follow. |
| LABEL | provides metada like maintainer. |
| ADD | Copies files and directories to the container. Can unpack compressed files. From url. |
| COPY | copies files and directories to the container.copy from local |
| **ARG** | Define build-time variable. |

**ANSIBLE**

Radically simple open source IT automation engine.

* Configuration management
* Provisioning
* Application Deployments
* Orchestration
* Uptime and site Reliability

**Ansible Terms**

* **Control Node:** Any Machine with Ansible installed.
* **Managed Nodes :** The network device(Servers) you manage with Ansible
* **Inventory :** A list of managed nodes.An inventory file is also sometimes called a 'hostfile'
* **Modules :** The units of code Ansible executes. Each modules has a particular functionality.
* **Tasks:** The Unit of action in Ansible.
* **Playbooks:** Ordered list of tasks.

LAB Setup

* Setup EC2 Instance
* Setup hostname
* create ansadmin user
* add user to sudoers file
* generate ssh keys
* Enable password based login
* install ansible

setup 3 AWS instances 1] Ansible control node 2] Tomcat 3] rhcl

Common Steps in all the instances

1] Setup hostname **/etc/hostname**

2] Create **ansadmin** user

useradd ansadmin

passwd ansadmin

3] add user to sudoers file to provide additional privileges to your user

**visudo**

Add into below tag

## Same thing without a password

# %wheel ALL=(ALL) NOPASSWD: ALL

ansadmin ALL=(ALL) NOPASSWD: ALL

4] Enable Password based authentication for **ansadmin** user

vi /etc/ssh/sshd\_config

# To disable tunneled clear text passwords, change to no here!

#PasswordAuthentication yes

5] Reload the configuration - **service sshd reload**

Below steps on Ansible control node server

Now establish connection between ansible server and Node

6] Generate ssh keys for ansadmin user

sudo su - ansadmin (/home/ansadmin)

(When we create new ec2 instance we download private key and in ec2-user in authentication key - public key)

**Normal Ec2 Instance login process**

User(Private Key) -- EC2 Instance Public Key(Authorization Key)

**In Ansible**

Ansible server (Private key) copy Public key into managed nodes

**ssh-keygen** (just hit enter)

It will generate private and public key (id\_rsa, id\_rsa.pub)

copy public key file from ansible to node - authoriation keys

ssh-copy-id ansadmin@172.31.17.192

7] **Install Ansible**

Check pythod is installed or not

**python --version**

**sudo yum install ansible** ( it will give new command to install ansible )

sudo amazon-linux-extras install ansible2

Check the ansible version **ansible --version**

Default installation loaction : **/etc/ansible**

1] Ad-hoc Command (No Idempotency)

**Ad-hoc commands** are commands which can be run indivitually to perform quick function

These ad-hoc commands are not used for configuration management and deployment because the commands are of one time usage

Adhoc commands (Simple Linux commands can execute)

ansible <group name>[o]

[0] Means first server of group

[-1] Means last server of group

ansible demo -ba "yum remove httpd -y"

**Modules (idempotency present)** by default first setup module execute- its gather the information about nodes

* Ansible Ships with a numbers of modules(Called Liabrary) that can be executed directly on remote hosts pr through playbooks
* ansible demo -b -m yum -a "name=httpd state=present" (for Install = Present , uninstall = absent and update = Latest)
* ansible demo -b -m yum -a "name=httpd state=latest"
* ansible demo -b -m service -a "name=httpd state=started"
* ansible demo -b -m user -a "name=raj"
* ansible demo -b -m copy -a "src=file4 dest=/tmp"
* ansible demo -m setup

**Playbook**

* Playbook are written in YAML Language (Yet another markup Language)
* It is human readable data
* Playbook is like a file where you write codes.Cosists of vars, tasks, handlers, files templates and roles.
* Each Playbook is compose of one or more modules in a list module is collection of configuration file
* Playbook are divided into 3 sections

1. Target Section: Define the host agaist which playbook task has to be executed
2. **Variable Section** Define variable
3. **Task Section :** List of all modules that we need to run in an order

**Handlers Section**

* A handler is exactly the same as a task, but it will run when called by another taks
* Handlers are just like regular tasks in an ansible playbook, but they are only run if the task contains a notify directive and also indicates that it changed something.

---

- hosts: demo

user: ansadmin

become: yes

connection: ssh

gather\_facts: yes

tasks:

- name: Install HTTPD on server

action: yum name=httpd state=installed

notify: restart httpd

handlers:

- name: restart httpd

action: service name=httpd state=restarted

**Ansible Vault**

* Ansible Allows keeping Sensitive data such as password or key in encrypted files rather than a plaintext in your playbooks
* **Creating a new encrypted playbook**

ansible-vault create vault.yml

* **Edit the encrypted playbook**

ansible-vault edit vault.yml

* **To change the password**

ansible-vault rekey vault.yml

* **To encrypt an existing playbook**

ansible-vault encrypt target.yml

* **To Decrypt an encrypted playbook**

ansible-vault decrypt target.yml

**Patching using Ansible playbook**

* Verify the application/database processes are running or not
* Decision point to start patching
* copy the required repo file to the managed hosts
* upgrade the kernel or package
* Check if reboot is required or not
* Reboot the system

host: patching

become:true

become\_user: root

tasks:

- name: Verify app/databse are running or not

shell: if ps -eaf | egrep 'httpd' | grep -v grep > /dev/null; then echo 'process running'; else echo 'process not running' ; fi

ignore\_errors: true

register: app\_proc\_check

debug: msg {{appr proc check.stdout}}

- name: Decition point to start patching

fail:

msg={{inventory\_hostname }} having running application,

when:app\_proc\_check.stdout== "process running"

-copy:

src:/home/kernel.repo

dest: /etc/yum.repos.d/

- name: current kernel version

shell: uname -r

register: curr\_ker

- name: initializing patching

yum:

name:kernel

state:latest

register:yum\_update

debug: msg="kernel has been updated"