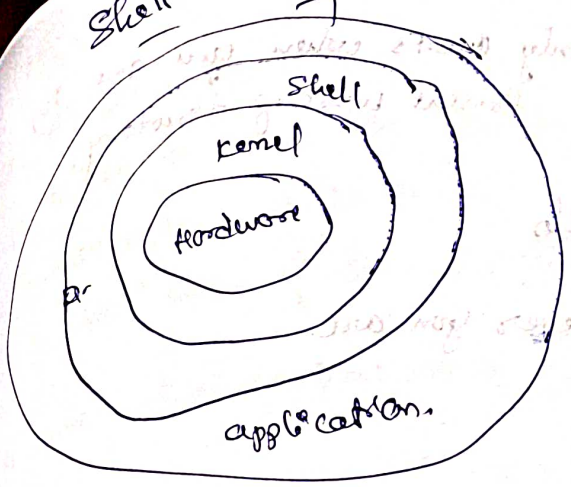
Shell scripting



shell
- bash
- sh
- ksh
- fish
- zsh

→ echo $0 → to check default shell
→ cat /etc/os-release → to check OS information

↳ echo $0
→ cat /etc/shells

↳ shell script consists of set of commands to perform a task. All commands execute sequentially.

↳ Some task like - file manipulation
- programs execution
- user interaction
- automation of tasks, etc

} can be done

Basic

#!/bin/bash
echo "Hello world"

} → Vi a1_basic.sh
use .sh to understand file is bash script.

#!/bin/bash → shebang

↳ make sure script has execute permission (rwx)

```
X X X
↓ ↓ ↘ others.
user group
```

↳ To run → ./filename.sh
(or)
→ bash filename } → for this permission not required
↳ chmod +x filename → to give permission.

To execute ./filename.sh

dot means pwd. we only works when you are in present working directory.

so, /path/of/the/filename.sh

To execute from wherever you are.

Comment line

1) Single line → using "#"

ex: #This is comment.

2) Multiline Comment

ex: <<comment

This is comment line

...

...

comment

Variables.

↳ Can store value or information & can be used multiple times

To define: VAR_NAME=VALUE

To print: echo $VAR_NAME

ex: #!/bin/bash

script to show how to use variables.

a=10

name="Prashant"

age=28

echo "My name is $name" and age is $age."

name="Chetan"

echo "My name is $name"

After execution ↳ My name is Prashant and age is 28

My name is Chetan.

VAR_NAME = $(hostname)

ex: #!/bin/bash

var to store the output of a command.

HOSTNAME = $(hostname)

echo "Name of this machine is $HOSTNAME"

↳ store value of hostname in HOSTNAME variable.

→ Once u define a variable and don't wanna change it until end of the script.

#!/bin/bash

constant variable

readonly COLLEGE="SAT"

echo "My college name is $COLLEGE"

Arrays

↳ if u want to store multiple values, how will u store?

↳ using arrays could be feasible.

How to define an array.

myArray=(7 20 30.5 Hello "Hey buddy")

How to get values from an array?

echo "${myArray[0]}"

Var name, index value

in above ex: index value of 7 is 0 → 2nd index value starts with zero.

20 is 1

30.5 is 2

Hello is 3

ex: #!/bin/bash

Array

myArray=(1 20 30.5 Hello "Hey buddy!")

echo "All the values in array are ${myArray[*]}"

echo "Value in 3rd index ${myArray[3]}".

How to get length of the array?

↳ echo "${#myArray[*]}"
array var name

ex:
#!/bin/bash
How to find no. of values in an array.
echo "No. of values (or) length of an array is ${#myArray[*]}"

How to get specific values? (i.e., range of values)

echo "${myArray[*]:1}"
↳ 2nd value

echo "${myArray[*]:2:3}"
starting from 2nd value → upto 3 values

ex: #!/bin/bash
How to get specific (or) range of values.
echo "values from index 2-3 ${myArray[*]:2:2}"
(From 3rd value get 2 values after this)

How to update an array?

~~echo~~ myArray+=(5 6 8)

ex:
#!/bin/bash
updating our array with new values
myArray+=(New 30 HO)
echo "Values of new array are ${myArray[*]}"

↳ Value of new array are 1 20 305 Hellow New 30 HO

Arrays Key-value

declare -A myArray.
myArray=([name]=chetan [age]=25 [city]=Blr)
echo "${myArray[name]}"

ex: #!/bin/bash
How to store the key values pairs
declare -A myArray
myArray=([name]=chetan [age]=25)
echo "Name is ${myArray[name]}"
echo "Age is ${myArray[age]}"

↳ Name is chetan
Age is 25.

String Operations

↳ used when u need to perform action after collecting some data/information/content. ex: slicing, caps to small.

myVar="Hello world!"
length = ${#myVar}
upper = ${value^^} → here value is Varname ex: myVar
lower = ${value,,}
replace = ${myVar/world/buddy}
slice = ${myVar:6:11}
From 7th character → upto 11 characters

ex:

```
#!/bin/bash
myVar = "Hey Buddy, How are you?"
myVarLength = ${#myVar}
echo "Length of the myVar is $myVarLength"

echo "Upper case is ---- ${myVar^^}"
echo "Lower case is ---- ${myVar,,}"

# To replace a string
newVar = ${myVar/Buddy/Chetan}
echo "New Var is ---- $newVar"
# To slice a string
echo "After slice ${myVar:4:5}"
```

User Interaction

Taking input from users

① read <var-name>

ex:
```
#!/bin/bash
echo "what is your name ?"
read name
echo "your name is $name"
```

→ 2nd here command line go to next line for input.

② read -p "your name" NAME

ex:
```
#!/bin/bash
read -p "what is your name ?" name
echo "your name is $name"
```

diff: here input u give in same line.

Arithmetic Operations

How to use expressions

let a++ (or) ((a++))
let a=5*10 ((a=5*10))

ex:
```
#!/bin/bash
# Maths Calculation
x=10
y=2
let mul = $x * $y
echo "$mul"
let sum = $x + $y
echo "$sum"
echo "Subtraction is $(($x - $y))"
```

Operators:

Equal ⟶ -eq / ==
Greater than or equal to ⟶ -ge
less than or equal to ⟶ -le
not equal ⟶ -ne / !=
Greater than ⟶ -gt
less than ⟶ -lt

Imp: -eq is for numerical comparisons
== is for string comparisons.
if -eq is used it will not consider string comparison strictly.

IF-ELSE

```
if [ condition ]
then
        echo " "
else
        echo " "
fi
```

ex:
```
#!/bin/bash
read -p "Enter your marks: " marks
if [[ $marks -gt 40 ]]   → space at starting & ending is important.
then
        echo "you are pass"
else
        echo "you are fail!!!!"
fi
```

ELIF condition → for multiple if condition

```
#!/bin/bash
read -p "Enter your marks: " marks
if [[ $marks -ge 80 ]]
then
        echo "1st division"
elif [[ $marks -ge 60 ]]
then
        echo "2nd division"
elif [[ $marks -ge 40 ]]
then
        echo "3rd division"
else
        echo "you are fail!!!"
fi
```

CASE

```
echo "Hey choose an option"
echo "a = To see the current date"
echo "b = list all the files in current dir"
read choise
case $choice in
    a) date ;;
    b) ls ;;
    *) echo "Non valid input"
esac
```

ex:
```
#!/bin/bash
echo "provide an option"
echo "a for print date"
echo "b for list of scripts"
echo "c for check the current location"

read choice

case $choice in
    a)
        echo "Today's date is:"
        date
        echo "Ending..."
        ;;
    b) ls ;;
    c) pwd ;;
    *) echo "Please provide valid value"
esac
```

LOGICAL OPERATORS (And, OR conditions)

① Condition 1 && Condition 2
↳ If both conditions are true → then → true, else false

② Condition 1 || Condition 2
↳ If any of the condition is true → then true

Ex:
```
#!/bin/bash
# AND operator
read -p "what is your age? " age
read -p "your country: " country.
if [[ $age -ge 18 ]] && [[ $country == "India" ]]
then
    echo "you can vote"
else
    echo "you can't vote"
fi
# OR operator
if [[ $age -ge 18 ]] || [[ $country == "India" ]]
then
    echo "you are allowed to India"
else
    echo "you are not allowed to India"
fi
```

③ Condition 1 && Condition 2 || Condition 3
↳ Execute Condition 2 only when 1 is true
else execute Condition 3.
↳ shortcut for if-else condition

Ex:
```
#!/bin/bash
# cond 1 && cond 2 || Cond 3
age=18
[[ $age -ge 18 ]] && echo "Adult" || echo "Minor"
```

FOR LOOP

```
for i in 1 2 3 4 5
do
    echo "Number is $i"
done.
```
↳ other ways to write for loop
```
for i in Raj Sham Baburao
for p in {1..20}   ↳ wild card.
```

Ex:
```
#!/bin/bash
for i in 1 2 3 4 5 6 7 8 9 10
do
    echo "Number is $i"
done
for name in Raju Sham Baburao
do
    echo "Name is $name"
done
for k in {1..5}   → instead of using 1 2 3 4 5
do
    echo "Numer is $i"
done
```

ITERATE VALUES FROM FILE (For loop using file)

```
items = "/home/[illegible]/PATH/of/file.txt"

for item in $(cat $items)
do
    echo $item
done.
```

ex:
```
#!/bin/bash
# Getting values from a file names.txt
FILE = "/home/myscripts/names.txt"
for name in $(cat $FILE)
do
    echo "Name is $name"
done.
```

For loop using Arrays

ex:
```
#!/bin/bash
myArray = ( 1 2 3 hello hi )
length = ${#myArray[*]}
for (( i=0; i<$length; i++ ))
do
    echo "value of array is ${myArray[$i]}"
done.
```

WHILE LOOP

similar to For, but while will loop until condition is true.

ex:
```
#!/bin/bash
count = 0
num = 10
while [[ $count -le $num ]]
do  echo "value of count var is $count"
    let count++
done.
```
↳ value of count is 1 2 ... 10

UNTIL LOOP

↳ just opposite to while, until condition becomes true it will stop until it will keep repeating loop.

ex: #!/bin/bash
```
a=10
until [[ $a -eq 1 ]]
do
    echo "value of a is $a"
    let a--
done
```

INFINITE LOOP

↳ used for constant monitoring, print, backup
↳ can use while or for

ex: Using while loop
```
#!/bin/bash
while true
do
    echo "Hi buddy"
done
```

using for loop
```
#!/bin/bash
for (( ;; ))
do
    echo "Hi buddy"
    sleep 2s
done
```

3) Create DNS records in your hosted zone

To read content from a file using while loop

```
while read myVar
do
    echo $myVar
done < file_name
```

→ create vi names.txt

Raju
Sham
Baburao

ex:
```
#!/bin/bash
while read myVar
do
    echo "Value from file is $myVar"
done < names.txt
```

To read content from a csv file using while loop

```
while IFS="," read f1 f2 f3
do
    echo $f1
    echo $f2
    echo $f3
done < file_name.csv
```

IFS → Internal field Separator

vi test.csv

```
id, name, age
01, Paul, 20
02, alex, 30
03, Raju, 40
```

ex:
```
#!/bin/bash
while IFS="," read id name age
do
    echo "Id is $id"
    echo "name is $name"
    echo "age is $age"
done < test.csv
```

Enhancement to above example using awk & while loop

ex:
```
#!/bin/bash
cat test.csv | awk 'NR=1 {print}' | while IFS="," read id name age
do
    echo "Id is $id"
    echo "Name is $name"
    echo "age is $age"
done
```

(NR → no. of row that is equal to 1; or NR!=1 → here 1st row i.e. id, name & age is removed during printing)

FUNCTIONS

- Block of code which perform some task and run when it is called.
- can be secure many times in our program which lessen our lines of code.
- we can pass arguments to the method.

To create

```
function myfun {
    echo "Hi"
}
```

(or)

```
myfun() {
    echo "Hello"
}
```

To call the function

```
myfun.
```

ex:
```
#!/bin/bash
function WelcomeNote {
    echo "------"
    echo "Welcome"
    echo "------"
}
#To call our function
WelcomeNote
WelcomeNote.
```

(or)

```
#!/bin/bash
# To make function
WelcomeNote() {
    echo "------"
    echo "Welcome"
    echo "------"
}
# To call our function
WelcomeNote
WelcomeNote
```

How to use Arguments in functions

```
addition() {
    local num1=$1
    local num2=$2
    let sum=$num1+$num2
    echo "sum of $num1 and $num2 is $sum"
}
```

↳ How to call → myfun 12 13

3) Create IAM records in your local pc.

ex: #!/bin/bash
# To create functions.
function welcome_note {
echo "- - - - -"
echo "welcome $1"
echo "Age is $2"
echo "- - - - -"
}
# To call functions & pass arguments.
welcome_note Raju 20
welcome_note Sham 30

ARGUMENTS IN SCRIPT

ex:
#!/bin/bash
to access the arguments
echo "First argument is $1"
echo "Second argument is $2"
echo "All the arguments are - $@"
echo "Number of arguments are - $#"
For loop to access the values from arguments.
for filename in $@
do
echo "Copying file - $filename"
done.

↳ here for loop will pick one by one & performs operations.

o/p: bash 26_args.sh test.csv myfile.csv
↳ copying file - test.csv
copying file - myfile.csv

SHIFTING ARGUMENTS

shift - when we pass multiple arguments, we can shift.

A BC
shift
BC

ex: #!/bin/bash
To create a user, provide username & description
echo "creating user"
echo "username is $1"
echo "Description is $2"

o/p: bash 27_file name chetan Test user, QA team
↳ username is Chetan
Description is Test

o/p, #!/bin/bash
echo "creating user"
echo "username is $1"
shift
echo "Description is $@"
↳ Username is chetan
Description is Test user, QA team.

3) Create DNS records in your hosted zone

OTHER USEFUL CONCEPTS

break → to stop the loop

ex:

```
#!/bin/bash
# example of break in a loop
# we just need to confirm if a given no. if present or not
no=6
for i in 1 2 3 4 5 6 7 8 9 10
do
    # Break the loop if no. found
    if [[ $no -eq $i ]]
    then
        echo "$no is found!!!"
        break
    fi
    echo "Number is : $i"
done
```

↳ Numbers is : 1
2
:
5
6 is found!!!

Continue → to stop current iteration of loop and start next iteration.

ex:

```
#!/bin/bash
# example of using continue in loop.
# suppose we only need to print odd no.
for i in 1 2 3 4 5 6 7 8 9 10
do
    let r=$i%2 ——→ modulus i.e. if no ÷ 2 = 0 o/p $0 not o/p
    if [[ $r -eq 0 ]]
    then
        continue
    fi
    echo "odd no. is $i"
done
```

↳ odd no. is : 1
— " — 3
— " — 5
— " — 7
— " — 9

sleep → to create delay between two executions
ex: sleep 1s → 1second.
1m → 1min.

exit → to stop script at a point

ex:

```
#!/bin/bash
#
if [[ $# -eq 0 ]]
then
    echo "please provide atleast one argument"
    exit 1
fi
echo "First argument is $1"
```

exit status ($?) → gives you status of previous command if that was successful.

ex:

```
#!/bin/bash
read -p "which site you want to check?" site
ping -c 1 $site
if [[ $? -eq 0 ]]
then
    echo "Successfully connected to $site"
else
    echo "Unable to connect $site"
fi
```

basename - strip directory info and only give filename

dirname - strip the filename and gives directory path

realpath - gives you full path for a file

↳ in all above 2, give file path → you get results
in realpath → give file → you get complete path.

CHECK IF FILE/DIR EXIST

if [-d folder_name] if folder exists
if [! -d folder_name] if folder not exists

if [-f file_name] if file exists
if [! -f file_name] if file not exists

eg:
```
#!/bin/bash
FILEPATH = "/home/chetan/myscripts/test.csv"
if [[ -f $FILEPATH ]]
then
    echo "FILE EXISTS"
else
    echo "Creating file now"
    touch $FILEPATH
fi
```

Random - A random integer between 0 and 32767 is generated.

eg:
```
#:/bin/bash
# Generating a random no. b/w 1 to 6
No = $[ $RANDOM%6 + 1 ])
echo "Number is $No"
```

UID → user ID of the user logged in

ex:
```
#!/bin/bash
# checking if the user is root or not
if [[ $UID -eq 0 ]]
then
    echo "User is root"
else
    echo "User is not root"
fi
```

Redirect (>) & append (>>)

eg:
```
#!/bin/bash
ping -c 1 www.goob.com >> redirect.log
```

DEV/NULL

In case if you don't wanna print the output of a command on terminal or write in a file, u can redirect the output to /dev/null
→ if u don't want to store them, redirect it to /dev/null

eg:
```
#!/bin/bash
read -p "which site you want to check" site
ping -c 1 $site &> /dev/null
```

Print name of the script

ex:
```
#!/bin/bash
echo "Name of the script is ${0}"
```

LOG MESSAGES

If you want to maintain the logging for your shell script you can use loggers in your script.

↳ you can find the logs under → /var/logs/messages

ex: # logger "Hey buddy!"

ex: #!/bin/bash
#example of logging
logger "This is log from $LOG"

DEBUGGING SCRIPTS

set -x → If we can enable the debugging of the script use

set -e → If we want to exit our script when a command fail.

ex: #!/bin/bash
set -e
pwd
date
cd /root
hostname.

AT - for scheduling only one time.

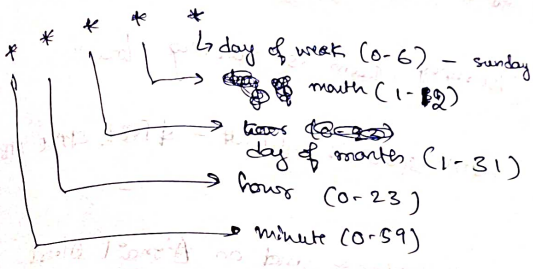
ex: → at 02:58 AM
↳ at> bash /home/chetan/34_redirect.sh
↳ at> ctrl + D

atq → To check scheduled jobs

atrm <id> → To remove the schedule

crontab
- crontab -l → To list crontab
- crontab -e → To create/edit crontab

ex: crontab -e
↳ * * * * * cd /home/chetan && ./create_file.sh



2) To check Free space

```
#!/bin/bash
FREE_SPACE=$(free -mt | grep "Total" | awk '{print $4}')
TH=500
if [[ $FREE_SPACE -lt TH ]]
then
        echo "Warning, Ram is running low"
else
        echo "RAM space is sufficient - $FREE_SPACE MB"
fi
```

2) Monitoring free Disk Space & send an Email Alert

Linux Boot loading

① Power on

② BIOS (Basic input/output system) → traditional uses MBR (MBR → Master Boot table) comes up with size Constraint upto 2TB
(or)
UEFI (Unified Extensible Firmware Interface)
↳ Modern one → uses GPT (GPT → Guide partition table) overcomes size constraint.
& its faster booting time compar to BIOS

↓ is loaded from non-volatile memory & these programs perform

POST (Power on self Test)
↓

③ Detects the devices connected to a system like CPU, RAM & storage & their connection & checkes everything is working fine
→ if problem sends "ERROR MESSAGES"
↓

④ chooses booting devices to boot the OS. (or) kernel function.
(this can be hard disk, server, CD ROM, (or) USB
↓ runs boot Loader

key jobs of boot loader are i) Locate the OS kernel in the disk
ii) load the kernel into the computer's memory
iii) Start running the kernel code

common used boot loaders GRUB (Grand unified Boot loader)
(or)
LILO (Linux loader) → outdated

(provides menu to choose the OS (or) kernel functions
↓ inserts linux kernel to memory & ~~starts~~ kernal starts up booting process.

*kernel

i) checks all the devices connected & starts Background processes

↓ next initial process just kicks-off

i.e. systemd.

↳ systemd uses target configuration files & decide which mode you should be booting into

↳ it manages the processes & services, probes all remaining hardware, mounts file system & runs a desktop environment.

↳ systemd activates default.target (runs target files)

other also activated units. like
↳ multi-user.target
↳ graphic.target

↓ systemd runs a set of startup scripts & configure the environment. (runs startup scripts)

users are presented with login window.

① User will 1st hit for DNS resolver
↳ User request i.e, DNS query to DNS resolver.
↳ DNS resolver intermediatory b/w User request & DNS name servers.

② DNS resolver checks for cache for i.p. adress for respective domain name.
↳ Cache depends on TTL (time-to-live)
↳ if TTL is more (or longer), DNS resolver DNS queries to Authoritative DNS i.e. Route 53
So charges will be less and if ttl is shorter charges will be more bcz. more DNS queries.

③ DNS resolver 1st checks with DNS root name server.
↳ next to TLD
↳ After getting this it forget to particular Route 53

How to set up DNS
↳ u can do it in Route 53 → Existing → update in registers.
↳ or other provider → New → hosted zone is created by Route 53 by itself

1) Register Domain name
2) Create a hosted zone in Route 53
3) Create DNS records in your hosted zone
↳ that will point traffic to specific resources.
4) Delegate to Route 53
↳ update the registrar with correct name servers.

↳ when u create hosted zone - 4 unique name records get created (ns-records)
↳ type : public

↳ point records at your server.
↳ Name
↳ TTC (seconds) ex: 300 sec
↳ value → i.p. address (or) ~~name~~

(or)

↳ Name
type : c-name [Canonical name]
↓
by not giving i.p. address.
↓
it gives other dns name.

↳ Alias → To attach resources i.p. address
target
↳ we don't want to hard code i.p. address → bcz dynamic
↳ Create Alias & point that to elastic load balancer (or) any L.B.
↳ select resource.

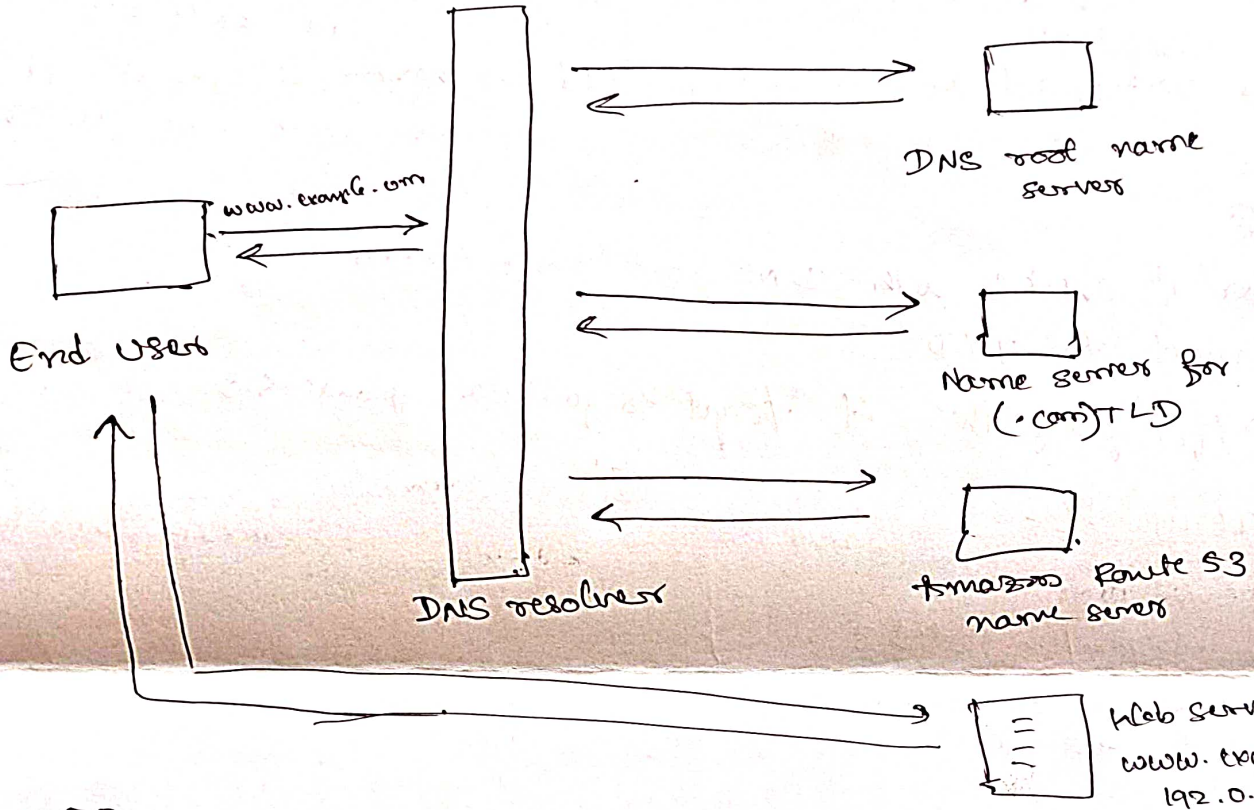
→ Route 53 supported AWS resources →

Create more records
1) MX record → for email service
2) TXT record → Certificates, email validation.

DNS

Domain Name system :→ Translates human readable domain name i.e., example.com to machine readable IP address i.e., 192.0.2.44

ex: Route 53 in AWS DNS Service.



Route 53

i) DNS name → website name

ii) Domain registrar

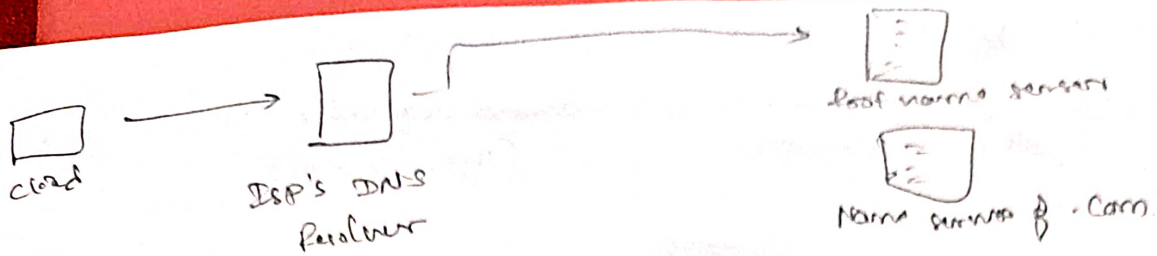
iii) Domain registry → company that sells domain w.r.t. TLD (Top-level Domain)
ex: .com .in

iv) TLD
Top level domain. ex: .com, .in, .co
a) Generic top-level-domain
b) Geographic top-level domain.

---

delegate
↳ u sets ns records created } → delegation set.
↳ delegate → changes can take upto to 48 hrs → bcz global cache for 48 hours

To check → dig example.com in linux
dig NS example.com ↳ to check / list of ns records.
dig NS example.com +trace.



→ DNS Resolver → ours by User's ISP → goes to nearest resolver
↳ looks for answer & returns back to client.

→ 1st it sends to Root name server
↳ only know last part i.e., .com (or) .in

→ if returns refers to other name server i.e .com

→ name server ~~knows~~ refers to Route53 for this to DNS resolver
↓
How it knows?

↳ when u register → registrar updates name to T.L.D & some information about who ur DNS provider

↳ Route 53 gives ip address

HOW Route 53

↳ when u create Route 53 - u create resource called hosted zone in Route 53
↓
Contains for all individual records (or) entries for that domain name & subdomain names

↳ Returns back to to web browser

DNS (Domain name system)

↳ DNS translates human readable names to machine readable IP addresses
(www.example.com) (192.0.2.44)

Types of DNS service

1) Authoritative DNS
   ↳ responsible for providing ip address to recursive DNS
   ↳ Amazon Route 53 is authoritative DNS.

2) Recursive DNS
   ↳ service also known as resolver
   ↳ stores DNS records temporarily (cached), but doesn't own's it.
   ↳ ~~If passes one or more time~~
   ↳ It sends sequence of requests to multiple authoritative DNS hence it's called recursive

How it works

1) User opens browser & enters website name ex: example.com
2) Request for example.com routed to DNS resolver (→ nearest resolver → take queries → returns to client), which is managed by user's ISP (Internet service provider) (like phonebook → check to pointers)
3) DNS resolver forwards request to DNS root name servers. (if not in cache)
4) DNS resolver forwards request to TLD (top-level-domain) ex: .com, .in which responds with Route 53 servers. name.
5) DNS resolver forwards request to those Route 53 servers & Route 53 responds with ~~Route~~ i.p. address of server to access that domain.
6) DNS resolver sends that i.p. address to browser & cache the it for some time. base on TTL (time-to-live) (billing based on ttl.)
7) The web browser sends the request to the i.p. address that got from the DNS resolver.
8) The web server ~~will~~ (s) other resource of the i.p. address returns the web page & web browser will display the page.