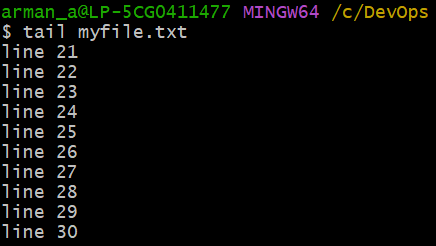
**LINUX**

**Files & Navigating:**

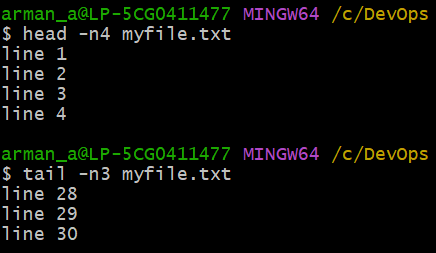
head: to show the top 10 lines of the file.



tail: to show the last 10 lines of the file.



Note: to show n number of lines of the file.



rm –rf dir-name: This command is used to delete directory(is not empty).

**System Info:**

Hostnamectl: To get information about os

|  |  |
| --- | --- |
| **Commands** | **Description** |
| [alias](https://www.javatpoint.com/linux-aliases) | Converts complex commands into simpler ones. |
| [bzip2 / bunzip2](https://www.javatpoint.com/linux-bzip2-bunzip2) | Compresses a file / Decompress a bzip2 file. |
| [bzcat / bzmore](https://www.javatpoint.com/linux-bzcat-bzmore) | Displays files compressed with bzip2. |
| [cal](https://www.javatpoint.com/linux-cal) | Displays calendar. |
| [cat](https://www.javatpoint.com/linux-cat) | It is a multi-function command. |
| [cd](https://www.javatpoint.com/linux-cd) | Changes the current working directory. |
| [chage](https://www.javatpoint.com/linux-user-password) | Sets an expiration date for a user account. |
| [chgrp](https://www.javatpoint.com/linux-file-ownership) | Changes group. |
| [chmod](https://www.javatpoint.com/linux-file-permissions) | Changes permission for a file or directory. |
| [chown](https://www.javatpoint.com/linux-file-ownership) | Changes the owner of a file or directory. |
| [chsh](https://www.javatpoint.com/linux-user-management) | Changes the shell. |
| clear | Clears the terminal. |
| [comm](https://www.javatpoint.com/linux-comm) | Compares two streams or files. |
| [cp](https://www.javatpoint.com/linux-cp) | Copies file content from one file to another file. |
| [cut](https://www.javatpoint.com/linux-cut) | Used to display the desired column from a file. |
| [date](https://www.javatpoint.com/linux-date) | Displays the current date. |
| [df](https://www.javatpoint.com/linux-df) | Checks the disk space in the system. |
| [echo](https://www.javatpoint.com/bash-read-user-input) | Prints the typed word on the terminal. |
| [exit](https://www.javatpoint.com/linux-exit-command) | Exits from the current user group to the last group. |
| [export](https://www.javatpoint.com/linux-export-command) | Exports shell variables to other shells. |
| [file](https://www.javatpoint.com/linux-file) | Displays the type of file. |
| [find](https://www.javatpoint.com/linux-find) | Finds files for a particular search. |
| [gpasswd](https://www.javatpoint.com/linux-groups) | Transfers group membership to another user. |
| [grep](https://www.javatpoint.com/linux-grep) | Filters lines of text containing a certain string. |
| [groupadd](https://www.javatpoint.com/linux-groups) | Creates a group. |
| [groupdel](https://www.javatpoint.com/linux-groups) | Permanently removes a group. |
| [groupmod](https://www.javatpoint.com/linux-groups) | Changes group name. |
| [groups](https://www.javatpoint.com/linux-groups) | Displays the group name to which the current user belongs to. |
| [gzip / gunzip](https://www.javatpoint.com/linux-gzip) | Compresses a file / Decompress a gzip file. |
| [head](https://www.javatpoint.com/linux-head) | Displays the first ten lines of a file. |
| [history](https://www.javatpoint.com/linux-shell-history-commands) | Displays older commands from the shell command history. |
| [HISTSIZE](https://www.javatpoint.com/linux-history-size) | Determines the number of commands to be stored in the current environment. |
| [HISTFILE](https://www.javatpoint.com/linux-history-size) | Displays the file that contains the history. |
| [HISTFILESIZE](https://www.javatpoint.com/linux-history-size) | Sets the number of commands kept in the history file. |
| [id](https://www.javatpoint.com/linux-id-command) | Tells about the user's id in the system. |
| [less](https://www.javatpoint.com/linux-less) | Displays file content according to the width of the terminal. |
| [locate](https://www.javatpoint.com/linux-locate) | Searches a file in the database. |
| [ls](https://www.javatpoint.com/linux-ls) | Lists all the files of a directory. |
| [man](https://www.javatpoint.com/linux-man) | Displays the manual page for the specified command. |
| [mkdir](https://www.javatpoint.com/linux-mkdir) | Creates directory. |
| [more](https://www.javatpoint.com/linux-more) | Displays one output screen at a time. |
| [mv](https://www.javatpoint.com/linux-mv) | Renames directories or files. |
| [od](https://www.javatpoint.com/linux-od) | Displays a file content in octal format. |
| [passwd](https://www.javatpoint.com/linux-user-password) | Set a password for a user group. |
| [pwd](https://www.javatpoint.com/linux-pwd) | Display the current working directory location. |
| [PS1](https://www.javatpoint.com/linux-ps) | Change the prompt name in the terminal. |
| [rename](https://www.javatpoint.com/linux-rename) | Renames more than one file at once. |
| [rm](https://www.javatpoint.com/linux-rm) | Removes a file. |
| [rmdir](https://www.javatpoint.com/linux-rmdir) | Removes a directory. |
| [set -o noclobber](https://www.javatpoint.com/linux-set-command) | Prevents file from getting overwritten. |
| [set +o noclobber](https://www.javatpoint.com/linux-set-command) | Allows overwriting in the existing file. |
| [set -u](https://www.javatpoint.com/linux-set-command) | Displays undefined variables as an error. |
| [set +u](https://www.javatpoint.com/linux-set-command) | Displays nothing for an undefined variable. |
| [set -x](https://www.javatpoint.com/linux-set-command) | Displays shell expansion. |
| [set +x](https://www.javatpoint.com/linux-set-command) | Disables shell expansion. |
| [sed](https://www.javatpoint.com/linux-sed) | Performs editing in streams. |
| [sleep](https://www.javatpoint.com/linux-sleep) | Waits for the specified number of seconds. |
| [sort](https://www.javatpoint.com/linux-sort) | sorts the content in alphabetical order. |
| [su](https://www.javatpoint.com/linux-su-commands) | It allows a user to run a shell as another user. |
| [sudo](https://www.javatpoint.com/linux-su-commands) | It allows a user to start a program with the credentials of another user. |
| [tac](https://www.javatpoint.com/linux-tac) | Displays file content in the opposite order. |
| [tail](https://www.javatpoint.com/linux-tail) | Displays the last ten lines of a file. |
| [tar](https://www.javatpoint.com/linux-gzip) | Compresses a directory. |
| [tee](https://www.javatpoint.com/linux-tee) | Puts stdin on stdout and then into a file. |
| [time](https://www.javatpoint.com/linux-time) | Displays time taken to execute a command. |
| [touch](https://www.javatpoint.com/linux-touch) | It creates an empty file. |
| [tr](https://www.javatpoint.com/linux-tr) | Translates characters. |
| [type](https://www.javatpoint.com/linux-shell-commands) | Displays information about command type. |
| [uniq](https://www.javatpoint.com/linux-uniq) | Sorts and display multi times repeating lines only once. |
| [unset](https://www.javatpoint.com/linux-set-environment-variable) | Removes a variable from a shell. |
| [useradd](https://www.javatpoint.com/linux-create-user) | Adds users. |
| [userdel](https://www.javatpoint.com/linux-user-management) | Deletes users. |
| [usermod](https://www.javatpoint.com/linux-groups) | Modifies the properties of a user. |
| [vi](https://www.javatpoint.com/vi-editor) | Opens vi editor to write a program. |
| [w](https://www.javatpoint.com/linux-users) | Displays who is logged on and what are they doing. |
| [wc](https://www.javatpoint.com/linux-wc) | Counts words, lines, and characters. |
| [who](https://www.javatpoint.com/linux-users) | Tells who is logged on the system. |
| [whoami](https://www.javatpoint.com/linux-users) | Tells the name of the user. |
| [who am i](https://www.javatpoint.com/linux-users) | Displays the line pointing to your current session. |
| [zcat / zmore](https://www.javatpoint.com/linux-zcat-zmore) | Views the files compressed with gzip. |

**Shell Scripting**

**What is shell scripting:** A shell script is a list of commands in a computer program that is run by the Unix shell which is a command line interpreter. A shell script usually has comments that describe the steps. The different operations performed by shell scripts are program execution, file manipulation and text printing. A wrapper is also a kind of shell script that creates the program environment, runs the program etc.

**What is shebang in shell scripting:** The Shebang “#!” The symbol indicates which interpreter, or which version of an interpreter, to use when running a script. It is also known as a "sharp-exclamation", "sha-bang", "hash-bang", or "pound-bang". The name is believed to have originated as a partial contraction of "SHarp bang" or "haSH bang"

Shell scripting variables:

* Local Variables
* Global Variables or Environment Variables
* Shell Variables or System Variables

## Local Variables:

A local variable is a special type of variable which has its scope only within a specific function or block of code. Local variables can override the same variable name in the larger scope.

#!/bin/sh

# Author: Arman Malik

# Variables in Shell scripting.

#simple example of variable.

NAME="Arman Malik"

echo "Hi, My name is $NAME"

#Concatenation of string with variable.

SPORT="Foot"

echo "My favorite sport is ${SPORT}ball."

**#Unsetting variables value.**

#Unsetting the value of variable using "unset" command.

Num=78

echo "Number is $Num"

unset Num

echo "Number is $Num" #it will not print the value of Num.

## Global Variables

A global variable is a variable with global scope. It is accessible throughout the program. Global variables are declared outside any block of code or function.

## Shell Variables

These are special types of variables. They are created and maintained by Linux Shell itself. These variables are required by the shell to function properly  They are defined in Capital letters and to see all of them, we can use set / env / printenv command. Below is the terminal shell pictorial depiction after executing the following command:-

Some useful shell variables are –

| **Variable Name** | **Description** | **Usage** |
| --- | --- | --- |
| BASH\_VERSION | Holds the version of this instance of bash. | echo $BASH\_VERSION |
| HOME | Provides a home directory of the current user. | echo $HOME |
| HOSTNAME | Provides computer name | echo $HOSTNAME |
| USERNAME | Provides username | echo $USERNAME |

**“expr" command:**

The **expr** command in Unix evaluates a given expression and displays its corresponding output. It is used for:

* Basic operations like addition, subtraction, multiplication, division, and modulus on integers.
* Evaluating regular expressions, string operations like substring, length of strings etc.

**Syntax:**

$expr expression

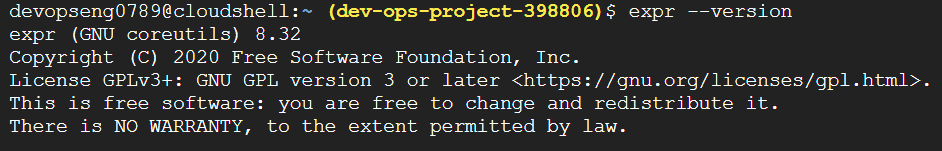
**Options:**

* **Option –version :** It is used to show the version information.

**Syntax:**

$expr --version

**Example:**

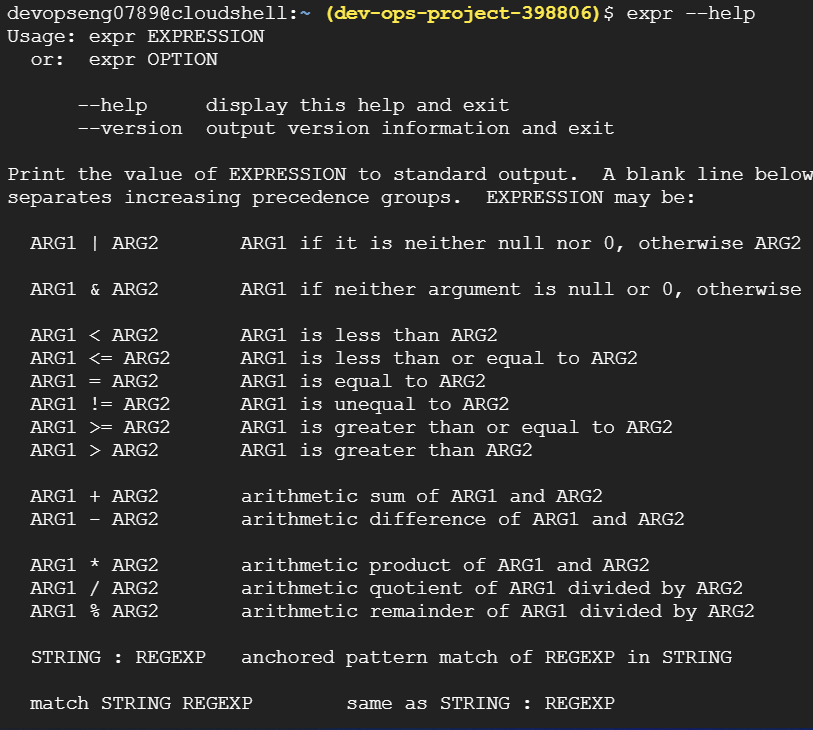


**Option --help :**It is used to show the help message and exit.

**Syntax:**

$expr –help

**Example:**

****

Below are some examples to demonstrate the use of “expr” command:

**1. Using expr for basic arithmetic operations :**

**Example:** Addition

$expr 12 + 8

**Output: 20**

**Example:** Multiplication

$expr 12 \\* 2

**Output: 24**

**Note:** The multiplication operator \* must be escaped when used in an arithmetic expression with expr.

**2. Performing operations on variables inside a shell script**

**Example:** Adding two numbers in a script

echo "Enter two numbers"

read x

read y

sum=`expr $x + $y` #x=2 and y=5

echo "Sum = $sum"

**Output: 7**

**Note:**expr is an external program used by Bourne shell. It uses expr external program with the help of backtick. The backtick(`) is actually called command substitution.

**3. Comparing two expressions**

#Comparing two expressions

x=10

y=20

# matching numbers with '='

res=`expr $x = $y`

echo $res

# displays 1 when arg1 is less than arg2

res=`expr $x \< $y`

echo $res

# display 1 when arg1 is not equal to arg2

res=`expr $x \!= $y`

echo $res

####  Evaluating boolean expressions   ####

# OR operation

$expr length "geekss"  "<" 5 "|" 19 - 6 ">" 10 #this is not working.

# AND operation

$expr length "geekss"  "<" 5 "&" 19 - 6 ">" 10 #this is not working.

#Finding length of a string

x=geeks

len=`expr length $x`

echo $len

#Finding substring of a string

x=geeks

sub=`expr substr $x 2 3`

#extract 3 characters starting from index 2

echo $sub

#Matching number of characters in two strings

expr geeks : geek

**“test” command:** A test command is a command that is used to test the validity of a command. It checks whether the command/expression is true or false. It is used to check the type of file and the permissions related to a file. Test command returns 0 as a successful exit status if the command/expression is true, and returns 1 if the command/expression is false.

**Syntax:**

$test [expressions]

**Example:**

a=12

b=0

if test "$a" -eq "$b" #or if test "$a" = "$b"

then

    echo "Hi"

else

    echo "bye"

fi

**Special NOTE:** Here, expression can be any command or expression that can be evaluated by the shell. And it is recommended to always enclose out test variables into double-quotes.

Here, are some of the operator flags that can be used with **test** command, along with their meaning:

**Flags for files and directories:**

* **test -e filename:** Checks whether the file exists or not. And return 1 if file exists and returns 0 if file does not exist.
* **test -d filename:** Checks whether the file is a directory or not. And returns 0 if the file is a directory and returns 1 if the file is not a directory.
* **test -f filename:**Checks whether the file is a regular file or not. And returns 0 if the file is a regular file and returns 1 if the file is not a regular file.
* **test -s filename:** Checks whether the file is empty or not. And returns 0 if the file is not empty and returns 1 if the file is empty.
* **test -r filename:** Checks whether the file is readable or not. And returns 0 if the file is readable and returns 1 if the file is not readable.
* **test -w filename:** Checks whether the file is writable or not. And returns 0 if the file is writable and returns 1 if the file is not writable.
* **test -x filename:**Checks whether the file is executable or not. And returns 0 if the file is executable and returns 1 if the file is not executable.

**Flags for  text strings**

* **string1 = string2:** Checks whether the two strings are equal or not. And returns 0 if the two strings are equal and returns 1 if the two strings are not equal.
* **string1 != string2:**Checks whether the two strings are not equal or not. And returns 0 if the two strings are not equal and returns 1 if the two strings are equal.
* **-n string:** Checks whether the string is empty or not. And returns 1 if the string is empty and returns 0 if the string is not empty.
* **-z string:** Checks whether the string is empty or not. And returns 0 if the string is empty and returns 1 if the string is not empty.

**Flags for comparison of numbers**

* **num1 -eq num2:** Checks whether the two numbers are equal or not. And returns 0 if the two numbers are equal and returns 1 if the two numbers are not equal.
* **num1 -ne num2:** Checks whether the two numbers are not equal or not. And returns 0 if the two numbers are not equal and returns 1 if the two numbers are equal.
* **num1 -gt num2:**Checks whether the first number is greater than the second number or not. And returns 0 if the first number is greater than the second number and returns 1 if the first number is not greater than the second number.
* **num1 -ge num2:** Checks whether the first number is greater than or equal to the second number or not. And returns 0 if the first number is greater than or equal to the second number and returns 1 if the first number is not greater than or equal to the second number.
* **num1 -lt num2:** Checks whether the first number is less than the second number or not. And returns 0 if the first number is less than the second number and returns 1 if the first number is not less than the second number.
* **num1 -le num2**: Checks whether the first number is less than or equal to the second number or not. And returns 0 if the first number is less than or equal to the second number and returns 1 if the first number is not less than or equal to the second number.

**Conditional flags**

* **condition1 -a condition2:**Checks whether the two conditions are true or not. And returns 0 if both the conditions are true and returns 1 if either of the conditions are false.
* **condition1 -o condition2:** Checks whether the two conditions are true or not. And returns 0 if either of the conditions are true and returns 1 if both the conditions are false.
* **!expression:**Checks whether the expression is true or not. And returns 0 if the expression is false and returns 1 if the expression is true.

**Command Substitution in Shell Scripting:**

#!/bin/sh

#Assigning a command to a variable

#ex 1

var=$(mkdir -p arman/malik)

echo $var

#ex 2

treeI=$(sudo apt install tree -y)

echo $treeI

echo "tree pkg Installed"

#ex 3

treeU=$(sudo apt remove tree -y)

echo $treeU

echo "tree pkg Uninstalled"

#storing output to a variable.

output=$(which git)

echo $output

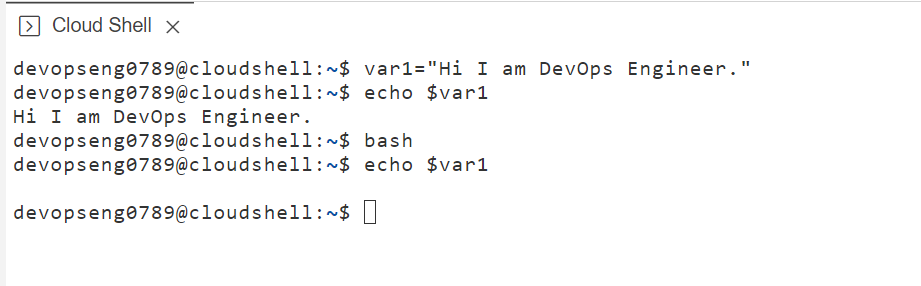
**Export Command:** To change the value of a global variable in a shell script, you can use the export command. The export command is used to mark **variables** and **functions** for export to the environment of subsequent commands. To declare a variable as global, you can simply declare it outside of any function or loop in your shell script.

If you declare variable var1 in parent process of linux but the scope is limit to the parent

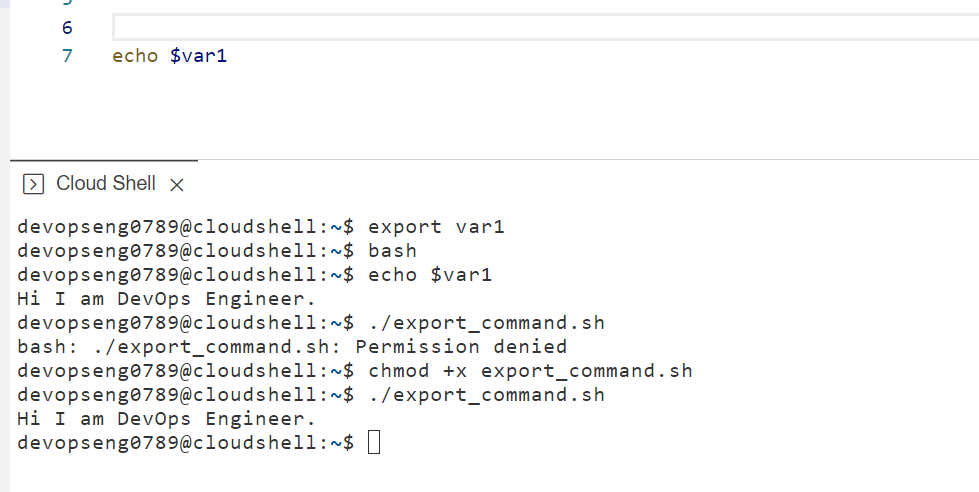
command you won't be able to access var1 in child process.

To access the var1 in child process, export command is used.

**Example:**

****

**Solution:**

****

**Command Line Arguments:**

#!/bin/bash

#arg start with the index of 0, but in this script we defined from 1, why?

# because 0th argument takes the file name.

#for example: bash myexample.sh arg1 aerg2 .....

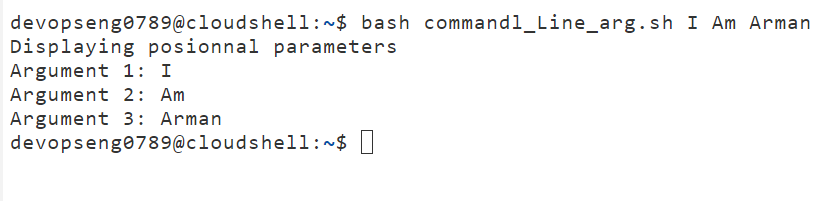
echo "Displaying positional parameters"

echo "Argument 1: $1"

echo "Argument 2: $2"

echo "Argument 3: $3"

**Output:**

****

**Conditional Statements:**

#!/bin/bash

#if and else statement.

val=10

if [ $val == 10 ]

then

    echo "Value is equal"

else

    echo "value is no equal"

fi

#if-elif-else statement

x=20

y=55

if [ $x == $y ]

then

    echo "x and y are equal"

elif [ $x \> $y ]

then

    echo "x is greater then y"

else

    echo "y is greater then x"

fi

#Nested if statement.

a=10

b=2

if [ $a == 10 ]

then

    if [ $b == 5 ]

    then

        echo `expr $a + $b`

    else

        echo `expr $a \\* $b`

    fi

else

    echo "cannot print anything"

fi

**Looping statements:**

#!/bin/bash

#while loop

a=1

# Iterating the loop until and less than 11.

while [ $a -lt 11 ]

do

    #printing the values.

    echo $a

    #incrementing the value of a by 1

    a=`expr $a + 1` #or ((a++))

done

#for loop

for a in 1 2 3 4 5 6 7 7 8 9 10

do

    echo $a

done

#until loop

a=0

# -gt is greater than operator

#Iterate the loop until a is greater than 10

until [ $a -gt 10 ]

do

    # Print the values

    echo $a

    # increment the value

    a=`expr $a + 1`

done

#break and continue statements.

Example of break

c=0

while [ $c -lt 10 ]

do

    if [ $c == 6 ]

    then

        break

    fi

    echo $c

    c=`expr $c + 1`

done

#Example of continue.

NUMS="1 2 3 4 5 6 7"

for NUM in $NUMS

do

   Q=`expr $NUM % 2`

   if [ $Q -eq 0 ]

   then

      #echo "Number is an even number!!"

      continue

   fi

   echo $NUM

done

# Example of Case statement.

echo "Which color do you like best?"

echo "1 - Blue"

echo "2 - Red"

echo "3 - Yellow"

echo "4 - Green"

echo "5 - Orange"

read color;

case $color in

  1) echo "Blue is a primary color.";;

  2) echo "Red is a primary color.";;

  3) echo "Yellow is a primary color.";;

  4) echo "Green is a secondary color.";;

  5) echo "Orange is a secondary color.";;

  \*) echo "This color is not available. Please choose a different one.";;

esac

**Functions:**#!/bin/bash

#creating a function

Hello()

{

    echo "I am DevOps Engineer"

}

#Invoking a function

Hello

#passing the parameters.

MyFun()

{

echo "I am $1 $2"

}

#Invoking a function

MyFun Arman Malik

#return statement in functions.

Person()

{

    echo "Hello $1 $2"

    return 25

}

#Invoking a function

Person Arman Malik

#capturing the return value

ret=$?

echo "You got ${ret}% discount"

#Nested Functions: calling one function in an another function.

First()

{

    echo "I am from 1st function"

}

Second()

{

    First # calling first() function inside second function.

    echo "I am from 2nd Function"

}

#Invoking second function.

Second

**Calling Function in another file:**

**Fun1.sh**

#!/bin/bash

add()

{

    echo "Adding Two numbers"

    echo `expr $1 + $2`

}

sub()

{

    echo "Subtracting two numbers"

    echo `expr $1 - $2`

}

**Fun2.sh**

#!/bin/bash

source fun1.sh

calculater()

{

    echo "Here the result of calculation"

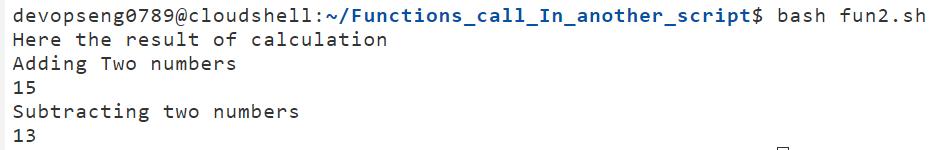
    add 10 5

    sub 20 7

}

calculater

**Output:**

**  
  
  
Editors:**

**NANO:** Nano is a user-friendly, simple and WYSIWYG(What You See Is What You Get) text editor, which improves the features and user-friendliness of UW Pico text editor. Unlike vim editor or any other command-line editor, it doesn’t have any mode. It has an easy GUI(Graphical User Interface) which allows users to interact directly with the text in spite of switching between the modes as in vim editor.

**#Installing Nano Text Editor**

1. Update the system

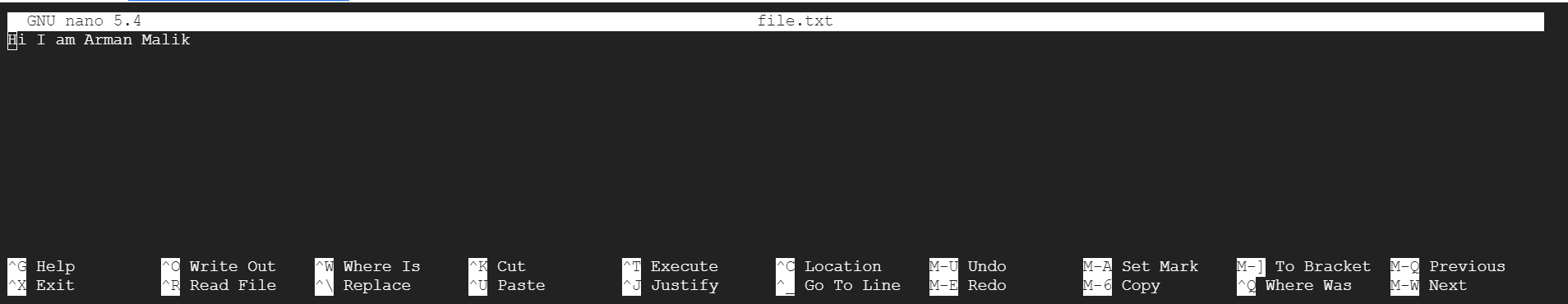
$sudo apt update

1. Install nano editor

$sudo apt install nano

1. Creating a file in nano editor.

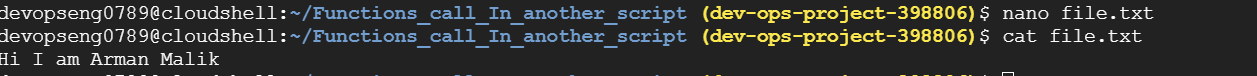
$nano new\_filename



1. Saving a file

press Ctrl+o

1. cat new\_filename



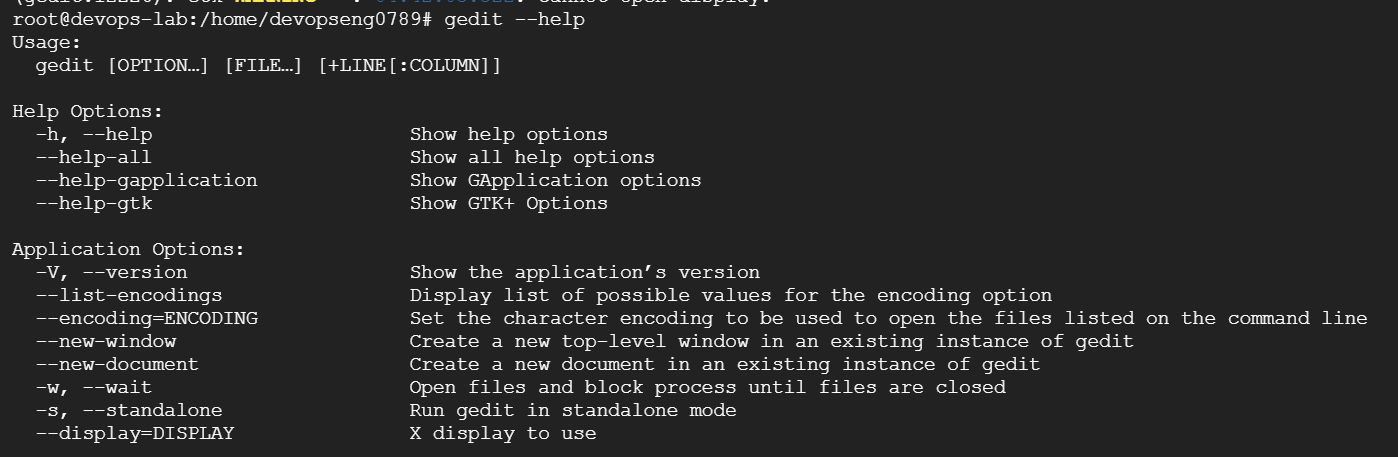
**PICO:** On [Unix-like](https://www.computerhope.com/jargon/u/unix-like.htm) operating systems, **pico** is a simple text [editor](https://www.computerhope.com/jargon/e/editor.htm) in the style of the [pine](https://www.computerhope.com/unix/upine.htm) e-mail composer.

**Tip:** On modern [Linux](https://www.computerhope.com/jargon/l/linux.htm) systems, **nano** (the [GNU](https://www.computerhope.com/jargon/g/gnu.htm) version of pico) is installed by default. It's identical to pico. In all the commands listed here, you can use "nano" instead of "pico" on Linux.

**GEDIT:** The gedit (GNOME Editor) is a flexible plugin for creating and editing files in Linux using the terminal. Other editor tools are available such as nano and vim, but gedit is preferred if the user wants to edit the file graphically. It is pre-installed by default in all the renowned Linux distributions. 

**Some Commands:**

* 1. gedit --help



**Advanced Commands in Shell Scripting:**

SED command in UNIX stands for stream editor and it can perform lots of functions on file like searching, find and replace, insertion or deletion. Though most common use of SED command in UNIX is for substitution or for find and replace. By using SED you can edit files even without opening them, which is much quicker way to find and replace something in file, than first opening that file in VI Editor and then changing it.

* SED is a powerful text stream editor. Can do insertion, deletion, search and replace(substitution).
* SED command in unix supports regular expression which allows it perform complex pattern matching.

**Syntax:**

**sed OPTIONS... [SCRIPT] [INPUTFILE...]**

**Example:**  
Consider the below text file as an input.

**$cat > geekfile.txt**

unix is great os. unix is opensource. unix is free os.

learn operating system.

unix linux which one you choose.

unix is easy to learn. unix is a multiuser os. Learn unix .unix is a powerful.

**Sample Commands**

1. **Replacing or substituting string :** Sed command is mostly used to replace the text in a file. The below simple sed command replaces the word “unix” with “linux” in the file.

**$sed 's/unix/linux/' geekfile.txt**

**Output :**

linux is great os. unix is opensource. unix is free os.

learn operating system.

linux linux which one you choose.

linux is easy to learn. unix is a multiuser os. Learn unix .unix is a powerful.

**Explanation:** Here the “s” specifies the substitution operation. The “/” are delimiters. The “unix” is the search pattern and the “linux” is the replacement string.

By default, the sed command replaces the first occurrence of the pattern in each line and it won’t replace the second, third…occurrence in the line.

1. **Replacing the nth occurrence of a pattern in a line :**Use the /1, /2 etc flags to replace the first, second occurrence of a pattern in a line. The below command replaces the second occurrence of the word “unix” with “linux” in a line.

**$sed 's/unix/linux/2' geekfile.txt**

**Output:**

unix is great os. linux is opensource. unix is free os.

learn operating system.

unix linux which one you choose.

unix is easy to learn. linux is a multiuser os. Learn unix .unix is a powerful.

1. **Replacing all the occurrence of the pattern in a line :**The substitute flag /g (global replacement) specifies the sed command to replace all the occurrences of the string in the line.

**$sed 's/unix/linux/g' geekfile.txt**

**Output :**

linux is great os. linux is opensource. linux is free os.

learn operating system.

linux linux which one you choose.

linux is easy to learn. linux is a multiuser os. Learn linux .linux is a powerful.

1. **Replacing from nth occurrence to all occurrences in a line :**Use the combination of /1, /2 etc and /g to replace all the patterns from the nth occurrence of a pattern in a line. The following sed command replaces the third, fourth, fifth… “unix” word with “linux” word in a line.

**$sed 's/unix/linux/3g' geekfile.txt**

**Output:**

unix is great os. unix is opensource. linux is free os.

learn operating system.

unix linux which one you choose.

unix is easy to learn. unix is a multiuser os. Learn linux .linux is a powerful.

1. **Parenthesize first character of each word :**This sed example prints the first character of every word in parenthesis.

**$ echo "Welcome To The Geek Stuff" | sed 's/\(\b[A-Z]\)/\(\1\)/g'**

**Output:**

(W)elcome (T)o (T)he (G)eek (S)tuff

1. **Replacing string on a specific line number :**You can restrict the sed command to replace the string on a specific line number. An example is

**$sed '3 s/unix/linux/' geekfile.txt**

**Output:**

unix is great os. unix is opensource. unix is free os.

learn operating system.

linux linux which one you choose.

unix is easy to learn. unix is a multiuser os. Learn unix .unix is a powerful.

The above sed command replaces the string only on the third line.

1. **Duplicating the replaced line with /p flag :**The /p print flag prints the replaced line twice on the terminal. If a line does not have the search pattern and is not replaced, then the /p prints that line only once.

**$sed 's/unix/linux/p' geekfile.txt**

**Output:**

linux is great os. unix is opensource. unix is free os.

linux is great os. unix is opensource. unix is free os.

learn operating system.

linux linux which one you choose.

linux linux which one you choose.

linux is easy to learn. unix is a multiuser os. Learn unix .unix is a powerful.

linux is easy to learn. unix is a multiuser os. Learn unix .unix is a powerful.

1. **Printing only the replaced lines :**Use the -n option along with the /p print flag to display only the replaced lines. Here the -n option suppresses the duplicate rows generated by the /p flag and prints the replaced lines only one time.

**$sed -n 's/unix/linux/p' geekfile.txt**

**Output:**

linux is great os. unix is opensource. unix is free os.

linux linux which one you choose.

linux is easy to learn. unix is a multiuser os. Learn unix .unix is a powerful.

If you use -n alone without /p, then the sed does not print anything.

1. **Replacing string on a range of lines :**You can specify a range of line numbers to the sed command for replacing a string.

**$sed '1,3 s/unix/linux/' geekfile.txt**

**Output:**

linux is great os. unix is opensource. unix is free os.

learn operating system.

linux linux which one you choose.

unix is easy to learn .unix is a multiuser os. Learn unix .unix is a powerful.

Here the sed command replaces the lines with range from 1 to 3. Another example is

**$sed '2,$ s/unix/linux/' geekfile.txt**

**Output:**

unix is great os. unix is opensource. unix is free os.

learn operating system.

linux linux which one you choose.

linux is easy to learn. unix is a multiuser os. Learn unix .unix is a powerful.

Here $ indicates the last line in the file. So the sed command replaces the text from second line to last line in the file.

1. **Deleting lines from a particular file :** SED command can also be used for deleting lines from a particular file. SED command is used for performing deletion operation without even opening the file.  
   1. To Delete a particular line say n in this example

**Syntax:**

$ sed 'nd' filename.txt

**Example:**

$ sed '5d' filename.txt

2. To Delete a last line

**Syntax:**

$ sed '$d' filename.txt

3. To Delete line from range x to y

**Syntax:**

$ sed 'x,yd' filename.txt

**Example:**

$ sed '3,6d' filename.txt

4. To Delete from nth to last line

**Syntax:**

$ sed 'nth,$d' filename.txt

**Example:**

$ sed '12,$d' filename.txt

5. To Delete pattern matching line

**Syntax:**

$ sed '/pattern/d' filename.txt

**Example:**

$ sed '/abc/d' filename.txt

**STTY Command: stty** command in Linux is used to change and print terminal line settings. Basically, this command shows or changes terminal characteristics.

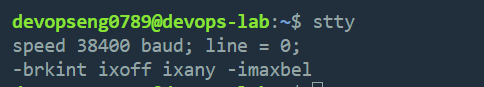
**Syntax:**

stty [-F DEVICE | --file=DEVICE] [SETTING]...

stty [-F DEVICE | --file=DEVICE] [-a|--all]

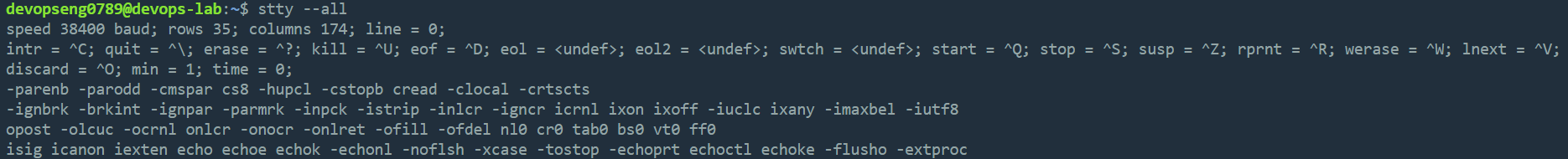
stty [-F DEVICE | --file=DEVICE] [-g|--save]

**Example:** It will display the characteristics of the terminal.



**Options:**

* **stty –all:**This option print all current settings in human-readable form.



* **stty -g:** This option will print all current settings in a stty-readable form.



* **stty -F :**This option will open and use the specified DEVICE instead of *stdin*.

**Example:**

stty -F D/

* **stty –help :** This option will display this help and exit.

stty --help

* **stty –version:** This option will show the version information and exit.

stty --version

**TOP Command: top** command is used to show the Linux processes. It provides a dynamic real-time view of the running system. Usually, this command shows the summary information of the system and the list of processes or threads which are currently managed by the Linux Kernel. As soon as you will run this command it will open an interactive command mode where the top half portion will contain the statistics of processes and resource usage. And Lower half contains a list of the currently running processes. Pressing *q*will simply exit the command mode.

*Here,*

* **PID:** Shows task’s unique process id.
* **PR:**The process’s priority. The lower the number, the higher the priority.
* **VIRT:** Total virtual memory used by the task.
* **USER:** User name of owner of task.
* **%CPU:**Represents the CPU usage.
* **TIME+:** CPU Time, the same as ‘TIME’, but reflecting more granularity through hundredths of a second.
* **SHR:** Represents the Shared Memory size (kb) used by a task.
* **NI:** Represents a Nice Value of task. A Negative nice value implies higher priority, and positive Nice value means lower priority.
* **%MEM:**Shows the Memory usage of task.
* **RES:** How much physical RAM the process is using, measured in kilobytes.
* **COMMAND:** The name of the command that started the process.

**Examples:**

**1) Exit Top Command After Specific repetition:**Top output keep refreshing until you press ‘q‘. With below command top command will automatically exit after 10 number of repetition.

top -n 10

**2) Display Specific User Process**

top -u paras

**3) Highlight Running Process in Top:** Press ‘z‘ option in running top command will display running process in color which may help you to identified running process easily.

**4) Shows Absolute Path of Processes:**Press ‘c‘ option in running top command, it will display absolute path of running pro.

**5) Kill running process:** You can kill a process after finding PID of process by pressing ‘k‘ option in running top command without exiting from top window as shown below.

**6) Sort by CPU Utilisation:**Press (Shift+P) to sort processes as per CPU utilization.

**7) Shows top command syntax:**

top -h

**8) Batch Mode :**Send output from top to file or any other programs.

top -b

**9) Secure Mode :**Use top in Secure mode.

top -s

**10) Command Line :** The below command starts top with last closed state.

Top -c

**11) Delay time :** It tells delay time between screen updates.

Top -d seconds.tenths

**AWK:** Awk is abbreviated from the names of the developers – **Aho, Weinberger, and Kernighan(AWK).**

Awk is a scripting language used for manipulating data and generating reports. The awk command programming language requires no compiling and allows the user to use variables, numeric functions, string functions, and logical operators.

Awk is a utility that enables a programmer to write tiny but effective programs in the form of statements that define text patterns that are to be searched for in each line of a document and the action that is to be taken when a match is found within a line. Awk is mostly used for pattern scanning and processing. It searches one or more files to see if they contain lines that matches with the specified patterns and then perform the associated actions.

**Syntax:**

awk options 'selection \_criteria {action }' input-file > output-file

**Sample Commands**

**Example:**

Consider the following text file as the input file for all cases below:

$cat > employee.txt

ajay manager account 45000

sunil clerk account 25000

varun manager sales 50000

amit manager account 47000

tarun peon sales 15000

deepak clerk sales 23000

sunil peon sales 13000

satvik director purchase 80000

**1. Default behavior of Awk:**By default Awk prints every line of data from the specified file.

$ awk '{print}' employee.txt

**Output:**

ajay manager account 45000

sunil clerk account 25000

varun manager sales 50000

amit manager account 47000

tarun peon sales 15000

deepak clerk sales 23000

sunil peon sales 13000

satvik director purchase 80000

In the above example, no pattern is given. So the actions are applicable to all the lines. Action print without any argument prints the whole line by default, so it prints all the lines of the file without failure.

**2. Print the lines which match the given pattern.**

$ awk '/manager/ {print}' employee.txt

**Output:**

ajay manager account 45000

varun manager sales 50000

amit manager account 47000

In the above example, the awk command prints all the line, which matches with the ‘manager’.

**3. Splitting a Line Into Fields :**For each record i.e line, the awk command splits the record delimited by whitespace character by default and stores it in the $n variables. If the line has 4 words, it will be stored in $1, $2, $3 and $4 respectively. Also, $0 represents the whole line.

$ awk '{print $1,$4}' employee.txt

**Output:**

ajay 45000

sunil 25000

varun 50000

amit 47000

tarun 15000

deepak 23000

sunil 13000

satvik 80000

In the above example, $1 and $4 represents Name and Salary fields respectively.

**Built-In Variables In Awk**

Awk’s built-in variables include the field variables—$1, $2, $3, and so on ($0 is the entire line) — that break a line of text into individual words or pieces called fields.

* **NR:** NR command keeps a current count of the number of input records. Remember that records are usually lines. Awk command performs the pattern/action statements once for each record in a file.
* **NF:** NF command keeps a count of the number of fields within the current input record.
* **FS:** FS command contains the field separator character which is used to divide fields on the input line. The default is “white space”, meaning space and tab characters. FS can be reassigned to another character (typically in BEGIN) to change the field separator.
* **RS:** RS command stores the current record separator character. Since, by default, an input line is the input record, the default record separator character is a newline.
* **OFS:** OFS command stores the output field separator, which separates the fields when Awk prints them. The default is a blank space. Whenever print has several parameters separated with commas, it will print the value of OFS in between each parameter.
* **ORS:** ORS command stores the output record separator, which separates the output lines when Awk prints them. The default is a newline character. print automatically outputs the contents of ORS at the end of whatever it is given to print.

**Examples:**

**Use of NR built-in variables (Display Line Number)**

$ awk '{print NR,$0}' employee.txt

**Output:**

1 ajay manager account 45000

2 sunil clerk account 25000

3 varun manager sales 50000

4 amit manager account 47000

5 tarun peon sales 15000

6 deepak clerk sales 23000

7 sunil peon sales 13000

8 satvik director purchase 80000

In the above example, the awk command with NR prints all the lines along with the line number.

**Use of NF built-in variables (Display Last Field)**

$ awk '{print $1,$NF}' employee.txt

**Output:**

ajay 45000

sunil 25000

varun 50000

amit 47000

tarun 15000

deepak 23000

sunil 13000

satvik 80000

In the above example $1 represents Name and $NF represents Salary. We can get the Salary using $NF , where $NF represents last field.

**Another use of NR built-in variables (Display Line From 3 to 6)**

$ awk 'NR==3, NR==6 {print NR,$0}' employee.txt

**Output:**

3 varun manager sales 50000

4 amit manager account 47000

5 tarun peon sales 15000

6 deepak clerk sales 23000

**More Examples**

**For the given text file:**

$cat > geeksforgeeks.txt

A B C

Tarun A12 1

Man B6 2

Praveen M42 3

**1) To print the first item along with the row number(NR) separated with ” – “ from each line in geeksforgeeks.txt:**

$ awk '{print NR "- " $1 }' geeksforgeeks.txt

1 - A

2 - Tarun

3 – Manav

4 - Praveen

**2) To return the second column/item from geeksforgeeks.txt:**

The question should be:- To return the second column/item from geeksforgeeks.txt:

$ awk '{print $2}' geeksforgeeks.txt

B

A12

B6

M42

**3) To print any non empty line if present**

$ awk 'NF < 0' geeksforgeeks.txt

here NF should be 0 not less than and the user have to print the line number also:

correct answer : awk ‘NF == 0 {print NR}’  geeksforgeeks.txt

OR

awk ‘NF <= 0 {print NR}’  geeksforgeeks.txt

0

**4) To find the length of the longest line present in the file:**

$ awk '{ if (length($0) > max) max = length($0) } END { print max }' geeksforgeeks.txt

13

**5) To count the lines in a file:**

$ awk 'END { print NR }' geeksforgeeks.txt

3

**6) Printing lines with more than 10 characters:**

$ awk 'length($0) > 10' geeksforgeeks.txt

Tarun A12 1

Praveen M42 3

**7) To find/check for any string in any specific column:**

$ awk '{ if($3 == "B6") print $0;}' geeksforgeeks.txt

**8) To print the squares of first numbers from 1 to n say 6:**

$ awk 'BEGIN { for(i=1;i<=6;i++) print "square of", i, "is",i\*i; }'

square of 1 is 1

square of 2 is 4

square of 3 is 9

square of 4 is 16

square of 5 is 25

square of 6 is 36

**GIT and GitHub**

* yum update –y : To update machine
* yum install git –y : To install git
* git --version: check the git version (my git version is 2.39.3)
* git init: to initialize the git.
* git status: to check the status of file.
* git add .: The git add command in general is used to add changed files in the current working directory to the staging area to tell git which files are going to be a part of the next commit.

The dot in the git add . command is simply a pathspec (which may also be a filepath), that tells git to only look for changed files in the current directory (i.e. it omits paths found in directories above and below).

* git commit –m “1st commit”: to commit
* git log: to see the logs
* git log --oneline: to see all logs in oneline
* git show: to see the changes in current file.
* git show <commit-id>: to see the changes in specific commit file.

***# How to push committed files on GitHub #***

**MY Github Token:** ghp\_MxOgeqFPomaWoWokOKux5JP0QOJ1Zp0gbRqB

Step 1: git remote add origin https://github.com/<user-name>/<repository-name>

Step 2: git remote set-url origin https:// <token-access-key>@github.com/<user-name>/<repository-name>

Ex: git remote set-url origin <https://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@github.com/arman-malik9/ArmanMalikPortfolio.github.io.git>

Step 3: git push origin master or main or branch-name

***# How to pull committed files from GitHub #***

**My Github Token:** ghp\_NCuZrJNdg5O1Elt8tBPgc48kDWfgi63fMa9y

Step 1: git remote add origin https://github.com/<user-name>/<repository-name>

Step 2: git remote set-url origin https:// <token-access-key>@github.com/<user-name>/<repository-name>

Ex: git remote set-url origin <https://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@github.com/arman-malik9/ArmanMalikPortfolio.github.io.git>

Step 3: git pull origin master or main or branch-name

***# Branches in Git #***

* git branch: To see the available branches
* git branch <branch-name>:To create a new branch
* git checkout <name of the branch you want to go>: To switch the branch
* git branch -d <branch-name>: To delete the branch
* git branch -D <branch-name>: To delete the branch forcefully
* git merge <branch-name>: To merge the branch with the master

**Git Conflict:** When same file having different content in different branches. If you merge, conflict occurs.

**How to resolve conflict?** After merging the branches, edit the conflict file as per your requirements, then add, and commit. No need to merge again.

**Git Stashing:** Stashing takes the dirty state of your working directory — that is, your modified tracked files and staged changes — and saves it on a stack of unfinished changes that you can reapply at any time (even on a different branch).

* git stash: To stash an item
* git stash list: To see stashed items list
* git stash apply stash@{0}: To get back item in workspace
* git stash clear: To clear the stashing area.

**Git Reset:** git reset is a powerful command that is used to undo local changes to the state of a git repo.

To reset staging area

git reset <filename>

git reset .

To reset the changes from both staging area and working directory at a time.

git reset –hard

**Git Revert:** when you write something wrong in the files and commit it, if you want to undo the operation on that files, git revert command is used.

git revert <commit-id> Then specify the comment “please ignore the previous commit”.

you get the revert file and now edit it.

**Note:**   
git reset is used before commit

git revert is used after commit

**How to remove untracked files:** Untracked files, those files which have never been add or commit or un-useful files.

git clean -n : (dry run) ask before deleting files

git clean -f : (forcefully) will not ask for yes or no for deleting

**Tagging:** Tag operation allows giving meaningful to a specific version in the repository.

* git tag –a <tag-name> -m <message> <commit-id> **:** To apply tag
* git tag : To see the list of tags
* git show <tag-name> : To see particular commit content by using tag
* git tag -d <tag-name> : To delete a tag

**Github Clone:** git clone <url-of github repo> : It creates a local repo automatically in linux m/c with the same name as in github account.

**Docker**

* To start docker

systemctl start docker

* To see all images present in your local machine

docker images

* To find out images in docker hub

docker search <image-name> eg: docker search jenkis or Ubuntu

* To download image from dockerhub to local machine

docker pull <image-name> eg: docker pull Ubuntu

* To create, run and give name to container

docker run -it --name <user-defined-container-name> <image-name> /bin/bash

eg: docker run -it --name my-ubuntu-image ubuntu /bin/bash

NOTE: run= create + start (-i means interactive mode and -t terminal)

* To check, service is started or not

service docker status

* To start container

docker start <user-defined-container-name> eg: docker start my-ubuntu-image

* To go inside container

docker attach <user-defined-container-name> eg: docker attach my-ubuntu-image

* To see all containers(running + stopped)

docker ps –a (ps means process state)

* To see only running containers

docker ps

* To stop container

docker stop <container-name>

* To delete container

docker rm <container-name>

* To delete image

docker rmi <your-image-id>

eg: docker rmi 9c7a54a9a43c

* To delete all images

docker rmi $(docker images -q)

**# Docker file component and ‘diff’ command #**

* LogIn into GCP and take ssh of compute engine instance
* Now we have to create Container from our own image.

Therefore, create one container first

docker run -it --name arman-container ubuntu /bin/bash

cd tmp/

* Now create one file inside this tmp directory

touch myfile

* Now if you want to see the difference between the base image & changes on it

docker diff arman-container

o/p: C /root

A /root/.bash\_history

C /tmp

A /tmp/myfile

NOTE: {C = change, A = append/addition, D = deletion}

* Now create image of this(arman-container) container

Docker commit <update-container> <new-image-name>

Eg: docker commit arman-container my-image

Check image: docker images

* Now create container from this image

Docker run -it –name malik-container my-image /bin/bash

* Now check tmp file in the container(malik-container)

Cd tmp/

Ls

o/p: myfile {you will get all files back}

**Docker file:**

Docker file is a text file, which contains some set of instruction.

OR

Automation of Docker image creation.

**Docker Component:**

FROM: for base image, this command must be on top of the docker file.

RUN: To execute commands, it will create a layer in image.

MAINTAINER: Author/Owner/Description (who create the docker file)

COPY: Copy file from local system (docker vm). We need to provide source, destination (we cannot download file from internet and any remote repo).

ADD: Similar to COPY but,it provide a feature to download files from internet, also we extract file at docker image side.

EXPOSE: TO expose points, such as post 8080 for tomcat/jenkis, port 80 for nginx etc.

WORKDIR: To set working directory for a container

CMD: Execute commands but during Container creation.

ENTERYPOINT: Similar to CMD but has higher priority over CMD, first command will be executed by ENTERYPOINT only.

ENV: Environment Variables.

ARG: **ARG** are also known as [build-time variables](https://docs.docker.com/engine/reference/builder/#arg). They are only available from the moment they are ‘announced’ in the Dockerfile with an ARG instruction up to the moment when the image is built. Running containers cannot access values of ARG variables.

Ex: Dockerfile

FROM ubuntu

MAINTAINER Arman malik <arman\_a@hcl.com>

RUN echo “I am DevOps Engineer” > /tmp/testfile

CMD [“echo”, “I am running container”]

To create image out of Dockerfile

docker build –t my-img .

docker ps –a

docker images

Now create container from the above image

docker run -it --name my-container my-img /bin/bash

**Simple view of creation of docker container**

Container

C

Container

B

Container

A

Dockerfile

Docker Image

**# Docker Volume & How to share it #**

* Volume is simply a directory inside our container.
* Firstly, we have to declare this directory as a Volume and then share volume.
* Even if we stop container, still we can access volume.
* Volume will be created in one container.
* You can declare a directory as a volume only while creating container.
* You can’t create volume from existing container.
* You can share one volume across any number of containers.
* Volume will not be included when you update an image.
* You can map volume in 2 ways.

Container 🡨🡪 Container

Host 🡨🡪 Container

**Benefits of Volume**

* Decoupling container from storage (means if container is deleted data stored in volume will not deleted)
* Share volume among different containers
* Attach volume to Containers
* On deleting container, volume will not delete.

**Creating Volume from Dockerfile**

FROM ubuntu

VOLUME ["/myVolume"]

1. Create Dockerfile 🡪
2. Create Image from the docker file

docker build –t img-vol

1. Create container from img-vol image. And check the volume named as “myVolume”

docker run –it –name vol-container img-vol /bin/bash

1. Now share Volume “myVolume” with another container

docker run -it --name vol-container2(new) --privileged=true --volumes-from vol-container ubuntu /bin/bash

1. Now you can create a file inside the volume “myVolume” in container “vol-container2”.

touch testfile.txt

1. Now exit from vol-container2

exit

1. Start vol-container

docker start vol-container

1. Go inside the vol-container

docker attach vol-container

1. Now check testfile.txt inside /myVolume

ls /myVolume/ {You will find the same file as you created in the vol-container2}

**Now try to create Volume by using Commands:**

docker run -it --name <container-name> -v /<volume-name> <image-name> /bin/bash

docker run -it --name container3 -v /volume2 ubuntu /bin/bash

ls

cd /volume2

Now create one file myfile-v2.0 and exit

Now create one more container, and share volume2

docker run -it --name container4 –privileged=true --volumes-from container3 ubuntu /bin/bash

Now you are inside container, do ls, you can see volume2 and its content.

**Some other commands related to docker volume:**

* docker volume ls : to see the list of volumes

**Example:**

[root@devops-lab docker-vol1.0]# docker volume ls

**DRIVER VOLUME NAME**

local 5093c9686a35b03a35760b85fd067c2cad0a924b7eff7f09ea47f8225e24e858

local 33964238dba2f9bbc189d71c464509c6c3999a992bdc99740d9129a1d0543e64

* docker volume create <volume-name> : to create the volumes
* docker volume rm <volume-name> : to remove the volume
* docker volume prune : It removes all unused docker volumes
* docker volume inspect <volume-name> : it shows the information about volume in the JSON format.

**Example:**

[root@devops-lab docker-vol1.0]# docker volume inspect 5093c9686a35b03a35760b85fd067c2cad0a924b7eff7f09ea47f8225e24e858

[

{

"CreatedAt": "2023-07-21T09:15:07Z",

"Driver": "local",

"Labels": {

"com.docker.volume.anonymous": ""

},

"Mountpoint": "/var/lib/docker/volumes/5093c9686a35b03a35760b85fd067c2cad0a924b7eff7f09ea47f8225e24e858/\_data",

"Name": "5093c9686a35b03a35760b85fd067c2cad0a924b7eff7f09ea47f8225e24e858",

"Options": null,

"Scope": "local"

}

]

* docker container inspect <container-name> : it shows the information about container in the JSON format.

**Docker Port Expose:** Means a user can access the container over the internet.

**Problem:** Container does not have IP address, so how to access Container over the internet?

**Answer:**  Instance has IP address, and we can assign the post number to container, then we can access the container over the internet.

i.e instance IP address = 34.131.181.133 intance also have 80 port number and Port number of Container = 80

Then, we can use as follow-> http://34.131.181.133:80

* service docker start : To start the docker service
* docker images : To see docker images
* docker run -td --name techserver -p 80:80 ubuntu : To create container and map port 80(instance) to 80(container). [-td means create container and run it but not get into the container.]

Another ex of port expose or mapping is 80:8080, 8080 is the default port number of Jenkins server.

* docker ps : To see the running containers
* docker port techserver : To check the mapping ports
* docker exec -it techserver /bin/bash : Get into the container
* apt-get update : update the container
* apt-get install apache2 : install apache server
* cd /var/www/html : change directory to html
* echo "checking for port mapping in docker" > index.html : creating and writing a html file
* ls : to see the files and directories
* cat index.html : check the contents of html file
* service apache2 start : to start the apache sever

**Difference between docker attach and docker exec**

Docker exc creates a new process in the container’s environment while docker attach just connect the standard Input/output of the main process inside the container to corresponding standard input/output error of current terminal.

Docker exc is specifically for running new things in a already started container, be it a shell or some other process.

Pid = process id

Ppid = parent process id

**Difference between expose and publish a docker**

Basically, we have 3 options:

1. Neither specify expose nor –p: If you specify neither expose nor –p the service in the container will only be accessible from inside container itself.
2. Only specify expose: If you expose a port the service in the container is not accessible from outside docker, but from inside other docker containers, this is good for inter-container communication.
3. Specify expose and –p: If you expose and –p a port, the service in the container is accessible from anywhere, even outside docker.

Note: if you specify –p but do not expose docker does an implicit expose. This is because if a pot is open to the public, it is automatically also open to the other docker containers.

Hence –p includes expose.

**How to push/pull docker images in dockerhub:**

docker images

docker run -it --name container-which-upload-to-dockerhub ubuntu /bin/bash

docker commit container-which-upload-to-dokcerhub image-from-container-arman

docker tag image-from-container-arman omararman/my-dir

docker login -u "myusername" -p "mypassword" docker.io

docker push myusername/myimage

docker pull myusername/myimage

**NOTE:**

1. Stop all running containers : docker stop $(docker ps –a -q)
2. Start all stopped containers : docker start $(docker ps –a -q)
3. Delete all stopped containers : docker rm $(docker ps –a -q)
4. Delete all images : docker rmi $(docker ps –a -q)

**Kubernetes**

**How to setup minikube on ubuntu machine**

sudo su

apt-get update

apt-get install apt-transport-https

apt install docker.io -y

docker --version

systemctl start docker

systemctl enable docker

sudo curl -s https://packages.cloud.google.com/apt/doc/apt-key.gpg | sudo apt-key add

nano /etc/apt/sources.list.d/kubernetes.list

apt-get update –y

apt-get install -y kubelet kubeadm kubectl kubernetes-cni

**BOOTSTRAPPING THE MASTER NODE (IN MASTER)**

kubeadm init

**COPY THE COMMAND TO RUN IN NODES & SAVE IN NOTEPAD**

mkdir -p $HOME/.kube

cp -i /etc/kubernetes/admin.conf $HOME/.kube/config

chown $(id -u):$(id -g) $HOME/.kube/config

kubectl apply -f raw.githubusercontent.com/coreos/flannel/master/Documentation/kube-flannel.yml kubectl apply -f https://raw.githubusercontent.com/coreos/flannel/master/Documentation/k8s-manifests/kube-flannel-rbac.yml

**CONFIGURE WORKER NODES (IN NODES)**

**COPY LONG CODE PROVIDED MY MASTER IN NODE NOW LIKE CODE GIVEN BELOW**

e.g- kubeadm join 172.31.6.165:6443 --token kl9fhu.co2n90v3rxtqllrs --discovery-token-ca-cert-hash sha256:b0f8003d23dbf445e0132a53d7aa1922bdef8d553d9eca06e65c928322b3e7c0

**GO TO MASTER AND RUN THIS COMMAND**

kubectl get nodes

**Kubernetes Objects:** Kubernetes uses Objects to represent the state of your cluster.

**The Basic Kubernetes Objects include:**

|  |  |
| --- | --- |
| * Pod * Service * Volume * Namespace * Replicasets | * Secrets * Configmap * Deployments * Jobs * Daemnonsets |

**Relationship between these objects:**

* Pod manages containers
* Replicaset manage pods.
* Deployment manages the Replicaset versions.
* Services expose pod processes to the outside world
* Configmap and secrets help you to configure pods.

**Simple Pod example:**

* kubectl apply -f podexample.yml

kind: Pod

apiVersion: v1

metadata:

 name: testpod

spec:

 containers:

  - name: c00

    image: ubuntu

    command: ["/bin/bash", "-c", "while true; do echo This is my 1st pod; sleep 8; done"]

* kubectl get pods
* kubectl get pods -o wide
* kubectl describe pods
* Kubectl describe pod <pod-name>
* kubectl logs -f firstpod
* Kubectl logs –f firstpod -c c00
* kubectl get nodes
* kubectl delete -f podexample.yml

or

* kubectl delete pod firstpod
* kubectl get cluster-info
* minikube addons list
* minikube dashboard
* minikube stop
* minikube delete
* minikube profile list
* kubectl get events

**Multi container Pod:**

* vi multicontainer-pod.yaml

kind: Pod

apiVersion: v1

metadata:

 name: multicontainer-pod

 annotations:

  description: This is the example of multicontainer pod

spec:

 containers:

  - name: c00

    image: ubuntu

    command: ["/bin/bash", "-c", "while true; do echo I am from container c00; sleep 8; done"]

  - name: c01

    image: centos

    command: ["/bin/bash", "-c", "while true; do echo I am from container c01; sleep 5; done"]

* kubectl apply –f multicontainer-pod.yaml
* kubectl logs -f multicontainer-pod : It gives the logs of container c00 only.
* kubectl logs -f multicontainer-pod -c c00: It gives the logs of container c00.
* kubectl logs -f multicontainer-pod -c c01: It gives the logs of container c01.
* kubectl logs -f multicontainer-pod -c c00 – hostname –I: gives pod IP 172.12.0.3
* kubectl logs -f multicontainer-pod -c c01 – hostname –I: gives same pod IP 172.12.0.3

**NOTE**: Because Containers do not have own IP, container c00 and c01 running on same pod, that is why both container gives same IP.

* kubectl exec multicontainer-pod -it -c c01 -- /bin/bash: To get into a specific container**.**

**Environment Variables in Pod:**

* vi env-var-pod.yaml

kind: Pod

apiVersion: v1

metadata:

 name: environments-variable-pod

 annotations:

  description: This is the example of env variable in pod..

spec:

 containers:

  - name: c00

    image: ubuntu

    command: ["/bin/bash", "-c", "while true; do echo This is the example of annotation; sleep 8; done"]

    env:

    - name: MYNAME

      value: Arman

* kubectl apply –f env-var-pod.yaml
* kubectl get pods
* kubectl exec environments-variable-pod -it -- /bin/bash
* env
* echo $MYNAME
* o/p -> Arman
* exit
* kubectl delete –f env-var-pod.yaml or kubectl delete pod environments-variable-pod

**Exposing port in Kubernetes:**

vi pod-with-ports.yaml

kind: Pod

apiVersion: v1

metadata:

 name: pod-with-ports

spec:

 containers:

  - name: c00

    image: httpd

    ports:

      - containerPort: 80

* kubectl apply –f env-var-pod.yaml
* kubectl get pods – o wide : it gives the pod IP and check it using given command that port is exposed or not.
* curl 10.244.0.24:80
* o/p It Works
* o/p -> Arman
* Now you can delete the pod.

**Labels and Selectors:**

Labels are the mechanism that use to organize kubernetes objects.

A label is key-value pair.

Multiple labels can be added to a single object.

* vi label.yaml

apiVersion: v1

kind: Pod

metadata:

  name: label-demo

  labels:

    environment: production

    app: nginx

spec:

  containers:

  - name: nginx

    image: nginx:1.14.2

    ports:

    - containerPort: 80

* kubectl get pods --show-labels: If you want see labels associated with pods.
* kubectl label pods label-demo myname=arman: If you want to add a label to an existing pod.

**Selectors:** There are 2 types of selectors

1. Equality based: (=,!=)
2. Set Based: (in, notin)

* Kubectl get pods -l environment=production: Gives a list of pod where ‘production’ label is present.
* Kubectl get pods -l environment!=production: Gives a list of pod where ‘production’ label is not present.
* kubectl get pods -l 'environment in (production, testing)': Gives the list of pods if environment variable matches any one of them.
* kubectl get pods -l 'environment notin (production, testing)': Gives the list of pods if environment variable does not match any one of them.
* kubectl delete pods -l 'environment notin (testing, development, production)': Delete all pods where above labels are not associated with pods.

**Node Selector**: One the application of label is used for selecting the node to create a pod to only be able to run on particular nodes.

* vi node-selector.yml

apiVersion: v1

metadata:

  name: nodelabels

  labels:

    env: development

spec:

    containers:

       - name: c00

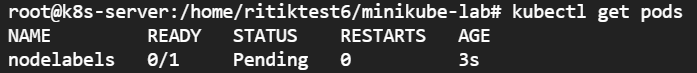
         image: ubuntu

         command: ["/bin/bash", "-c", "while true; do echo Hello-world; sleep 5 ; done"]

    nodeSelector:

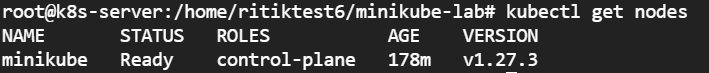
       hardware: t2-medium

* kubectl apply --f node-selector.yml
* kubectl get pods

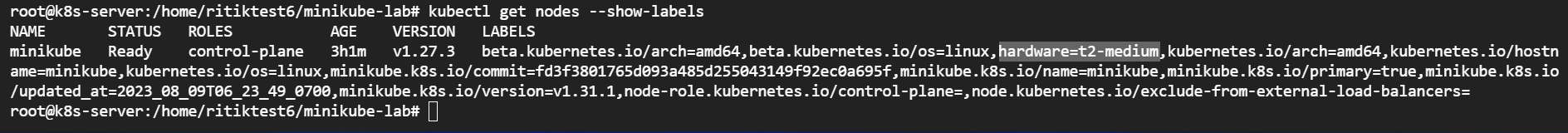


We will get the pending status, because we did not define the labels on nodes. So lets define the label on node.

* Kubectl get nodes

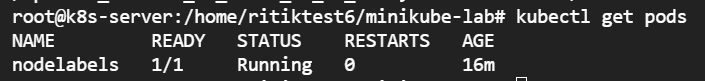


* kubectl label nodes minikube hardware=t2-medium
* kubectl get nodes --show-labels



Now we can check the pod is in running state.

* kubectl get pods



**How to un-label or remove label from pod/node?**

We used command for labeling: kubectl label pod <pod-name> name=arman

To remove label use the following command: kubectl label pod <pod-name> name-

**Scaling and Replication in Kubernetes:** By default, kubernetes does not have the features of Scaling and Replication, but with the help of some API and object, we can add it.

**Use of ReplicationController Object:** maintain the current state with desired state of the pods.

kind: ReplicationController

apiVersion: v1

metadata:

  name: myreplica

spec:

  replicas: 2

  selector:

    myname: Arman Malik

  template:

    metadata:

      name: testpod6

      labels:

        myname: Arman

    spec:

     containers:

       - name: c00

         image: ubuntu

* kubectl get pods
* kubectl get rc
* kubectl describe rc <replicationController-name>
* kubectl scale --replicas=5 rc -l myname=arman
* kubectl label rc myreplica -l city=Noida: Apply the label
* kubectl get rc --show-labels
* kubectl scale --replicas=0 rc -l city=Noida: we can have 0 pods in rc also.
* kubectl delete rc myreplica: Deleting the ReplicationController

**Replica Set:** Replica set is much similar to Replication Controller object, the main difference is Replication Controller works only equality based Selectors while Replica Set works on both equality based and set-based selectors.

**NOTE**: Replica Set is not available in apiVersion v1, but available in apiVersion apps/v1

kind: ReplicaSet

apiVersion: apps/v1

metadata:

  name: myrs

spec:

  replicas: 2

  selector:

    matchExpressions:                             *# these must match the labels*

      - {key: myname, operator: In, values: [arman, malik, omar]}

      - {key: env, operator: NotIn, values: [production]}

  template:

    metadata:

      name: testpod7

      labels:

        myname: malik

        env: development

    spec:

     containers:

       - name: c00

         image: ubuntu

         command: ["/bin/bash", "-c", "while true; do echo I am DevOps Engineer; sleep 5 ; done"]

* kubectl get pods
* kubectl get rs –o wide
* kubectl scale --replicas=5 rs/myrs
* kubectl delete rs myrs

**Deployment:** A deployment provides declarative updates for pods through replicaset.

Example:

vi deployment.yml

kind: Deployment

apiVersion: apps/v1

metadata:

   name: mydeployments

spec:

   replicas: 2

   selector:

    matchLabels:

     name: deployment

   template:

     metadata:

       name: testpod

       labels:

         name: deployment

     spec:

      containers:

        - name: c00

          image: ubuntu

          command: ["/bin/bash", "-c", "while true; do echo Welcome to DevOps; sleep 5; done"]

* kubectl get pods:
* kubectl get deploy or kubectl get deployment or kubectl get deployments: will give same output.
* Kubectl describe deploy mydeployments
* Kubectl get rs
* Kubectl scale --replicas=3 deploy mydeployments

Now make some changes in deployment.yml and apply.

* Kubectl apply –f deployment.yml
* Kubectl get pods
* Kubectl get deploy
* kubectl rollout status deployment mydeployments
* kubectl rollout history deployment mydeployments
* kubectl rollout undo deployment mydeployments
* kubectl delete deploy mydeployments

**Kubernetes Networking**

**Communication between containers, which are in the same pod:**

* vicontainer-communication.yml

kind: Pod

apiVersion: v1

metadata:

  name: testpod

spec:

  containers:

    - name: c00

      image: ubuntu

      command: ["/bin/bash", "-c", "while true; do echo Hello-World; sleep 5 ; done"]

    - name: c01

      image: httpd

      ports:

       - containerPort: 80

* Kubectl apply –f container-communication.yml
* Kubectl get pods
* Kubectl exec testpod –it –c c00 -- /bin/bash
* Apt update && apt install curl –y
* Curl localhost:80

o/p: It works

**Pod-to-Pod Communication, within the same node:** Every pod has its own IP address they can communicate with IP.

**NOTE**: By default, Pod IP cannot be accessible outside the node.

* vi pod1.yml

kind: Pod

apiVersion: v1

metadata:

 name: pod1

spec:

 containers:

  - name: c01

    image: httpd

    ports:

     - containerPort: 80

* Kubectl apply –f pod1.yml
* Vi pod2.yml

kind: Pod

apiVersion: v1

metadata:

 name: pod2

spec:

 containers:

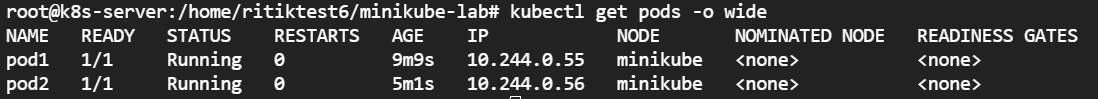
  - name: c02

    image: nginx

    ports:

     - containerPort: 80

* Kubectl apply –f pod2.yml
* Kubectl get pods -o wide



* Kubectl exec –it pod1 -- /bin/bash
* Apt update && apt install curl
* Curl 10.244.0.56:80
* O/P: Welcome
* Exit
* Kubectl exec –it pod2 -- /bin/bash
* Apt update && apt install curl
* Curl 10.244.0.55:80
* O/P: It works
* Exit

**‘Service’ Object in K8S:**

**Problem:** We know that if pod is damage, then new pod is created every time and ReplicaSet maintain it.

The pod or pods can be created within the same node or in different node within the same cluster.

The new pod gets always different IP address, so it’s hard to remember new IP address of the pod every time to communicate with other pod.

Service is a logical bridge between pods and end-users, which provides virtual IP (VIP).

**Understanding How Service object works**

Virtual IP: 127.10.4.12

POD 4

POD 3

POD 2

POD 1

Container

Container

Container

Container

NODE B

NODE A

ReplicaSet/ReplicationController

Service

**Cluster**

The Virtual IP of provided by service object is mapped to pod IPs, now instead of, for communication of pods using pods IP, we use Virtual IP which will never change.

Kube-proxy is the one, which keeps the mapping between VIP and pods upto date.

Note: The pod cannot be accessed outside the cluster but Service object allow to access of pods outside the cluster.

Labels are used to select which are the pods to be put under a Service.

By default, Service can only run between ports in the range 30,000-32,767.

Service are four Types:

1. ClusterIP (default)
2. NodePort
3. LoadBalancer
4. Headless

* Vi deploy-pod.yml

kind: Deployment

apiVersion: apps/v1

metadata:

   name: mydeployments

spec:

   replicas: 1

   selector:      *# tells the controller which pods to watch/belong to*

    matchLabels:

     name: deployment

   template:

     metadata:

       name: testpod1

       labels:

         name: deployment

     spec:

      containers:

        - name: c00

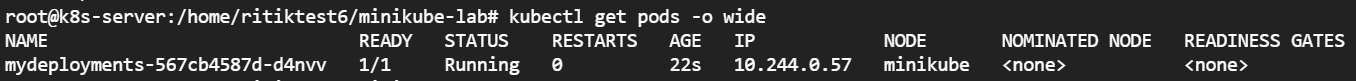
          image: httpd

          ports:

          - containerPort: 80

Kubectl apply –f deploy-pod.yml

Kubectl get pods –o wide



* Curl 10.244.0.57

O/P: It works

* Vi service.yml

kind: Service                             *# Defines to create Service type Object*

apiVersion: v1

metadata:

  name: demoservice

spec:

  ports:

    - port: 80                               *# Containers port exposed*

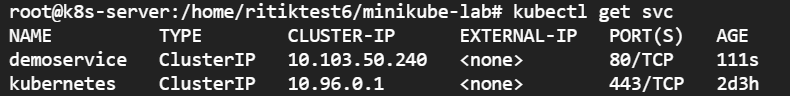
      targetPort: 80                     *# Pods port*

  selector:

    name: deployment                    *# Apply this service to any pods which has the specific label*

  type: ClusterIP                       *# Specifies the service type i.e ClusterIP or NodePort*

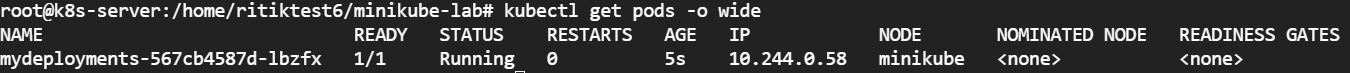
* kubectl apply –f service.yml
* kubectl get svc



* Curl 10.103.50.240 Cluster Virtual IP

O/P It works Same output

* Kubectl delete pod mydeployments-567cb4587d-d4nvv
* Kubectl get pods –o wide



Now the Pod IP is change after recreating 10.244.0.57 to 10.244.0.58,

But must access this pod using VIP only.

* Kubectl get deploy
* kubectl delete deploy mydeployments
* kubectl get pods
* kubectl delete service demoservice
* kubectl get svc

**NodePort:** As we know that ClusterIP can be used for within a node, but If we want to access pod outside cluster or using internet then service type NodePort comes into the picture.

In short, NodePort makes a service accessible from outside the cluster.

**How NodePort works:**  Service type NodePort gives the VIP and as well as a port number between 30,000 to 32,767.

Using node IP address + NodePort number we can access the service.

node IP address + NodePort number 34.162.188.198:31320

Cluster VIP (check which pod is mapped to the 31320 port)

Mapped to Cluster VIP Response

POD 2 port: 8080

POD 1 port: 80

Mapped 31320:80 Response

* Vi pod.yml

kind: Deployment

apiVersion: apps/v1

metadata:

   name: mydeployments

spec:

   replicas: 1

   selector:      *# tells the controller which pods to watch/belong to*

    matchLabels:

     name: deployment

   template:

     metadata:

       name: testpod1

       labels:

         name: deployment

     spec:

      containers:

        - name: c00

          image: httpd

          ports:

          - containerPort: 80

* Kubectl apply –f pod.yml
* Kubectl get pods
* Vi service.yml

kind: Service                             *# Defines to create Service type Object*

apiVersion: v1

metadata:

  name: demoservice

spec:

  ports:

    - port: 80                               *# Containers port exposed*

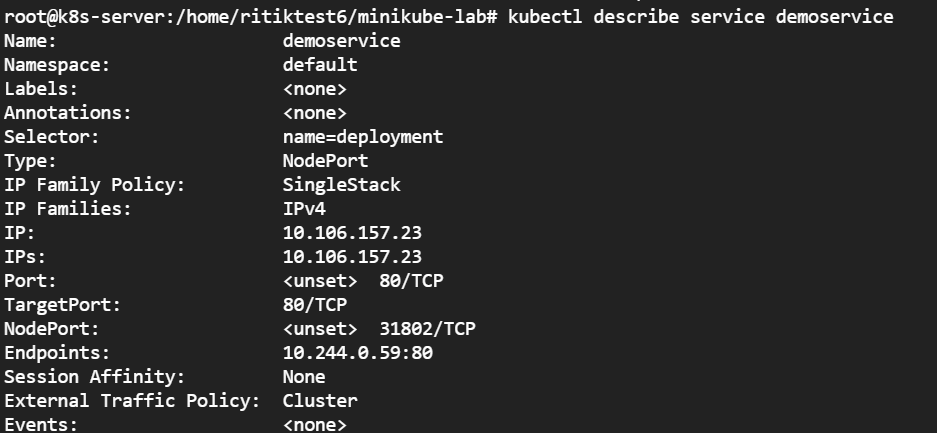
      targetPort: 80                     *# Pods port*

  selector:

    name: deployment                    *# Apply this service to any pods which has the specific label*

  type: ClusterIP                       *# Specifies the service type i.e ClusterIP or NodePort*

* kubectl apply –f service.yml
* kubectl get svc
* kubectl describe demoservice



**Volumes:**

* Containers are short lived in nature. All data stored inside a container is deleted if the container is crashes.
* To overcome this problem, kubernetes uses volume.
* A volume us essentially a directory backed by a storage medium and it content are determined by the volume type.
* In kubernetes, a volume is attached to a pod and shared among the containers of that pod.

The volume has the same life span as the pod.

**Volume Types:**

A volume type decides the properties of the directory, like size, content etc.

Some example of volume types are-

1. Node-local type such as **emptydir** and **hostpath**.
2. File sharing types such as **nfs.**
3. Cloud provider-specific type like **awselasticblockstore**, **azuredisk**.
4. Distributes system types, **glusterfs** or **cephfs**.
5. Special purpose types like **secret**, or **gitrepo**.

**EmptyDir:**

* Use this when we want to share contents between multiple containers on the same pod and not to the host machine.
* An emptydir volume is first created when a pod is assigned to a node and exist as long as that pod is running on that node.
* As the name says, it is initially empty.
* Containers in the pod can all read and write the same files in the emptydir volume. Though that volume can be mounted at the same or different paths in each containers.
* A container crashing does not remove a pod from a node; the data in an emptydir volume is safe across container crashes.

POD

Volume

Container B

Container A

* vi emptydir-pod.yml

apiVersion: v1

kind: Pod

metadata:

  name: myvolemptydir

spec:

  containers:

  - name: c1

    image: centos

    command: ["/bin/bash", "-c", "sleep 15000"]

    volumeMounts:                                    *# Mount definition inside the container*

      - name: xchange

        mountPath: "/tmp/xchange"

  - name: c2

    image: centos

    command: ["/bin/bash", "-c", "sleep 10000"]

    volumeMounts:

      - name: xchange

        mountPath: "/tmp/data"

  volumes:

  - name: xchange

    emptyDir: {}

* kubectl apply –f emptydir-pod.yml
* kubectl get pods
* kubectl exec -it myvolemptydir -c c1 -- /bin/bash
* cd /tmp/xchange
* touch myfile.txt
* ls

o/p = myfile.txt

* exit
* kubectl exec -it myvolemptydir -c c1 -- /bin/bash
* cd /tmp/data
* ls

o/p = myfile.txt

**Hostpath:**

* Use this when we want to access the content of a pod/container from hostmachine.
* A hostpath volume mounts a file or directory from the host node’s filesystem into your pod.
* Also, hostpath volume is shared among containers running in that pod.

POD

*Mounted*  
  
  
  
  
 **/tmp/data**

HOST

Volume

Container B

Container A

* Vi hostpath-pod.yml

apiVersion: v1

kind: Pod

metadata:

  name: myvolhostpath

spec:

  containers:

  - image: debian

    name: testc

    command: ["/bin/bash", "-c", "sleep 15000"]

    volumeMounts:

    - mountPath: /tmp/hostpath

      name: testvolume

  volumes:

  - name: testvolume

    hostPath:

      path: /tmp/data

* kubectl apply –f hostpath-pod.yml
* kubectl get pods
* kubectl exec -it myvolhostpath -- /bin/bash
* cd /tmp/hostpath
* touch myfile.txt
* ls

o/p = myfile.txt

* exit
* cd /tmp/data
* ls

o/p = myfile.txt

**Persistent Volume & Liveness Probe (Healthcheck):**

**Persistent Volume:** It means the volume is always persistent if running pod crashed and new pod is created, you will find same data in new pod as it was in the crashed pod.

POD A

us

POD C

POD B

**Persistent Volume Claim**

**Persistent Volume**

**Steps for creating persistent volume and used in a pod:**

Disk

**Step 1:**  Create a disk in GCP by using the following command.

gcloud compute disks create --size=500GB --zone=us-east5-a my-data-disk

**Step 2:** Create persistent volume.

* vi pv.yml

apiVersion: v1

kind: PersistentVolume

metadata:

  name: test-volume

spec:

  capacity:

    storage: 400Gi

  accessModes:

  - ReadWriteOnce

  gcePersistentDisk:

    pdName: my-data-disk *# must be created in gcp disk before specified here.*

    fsType: ext4

* kubectl apply -f pv.yml
* kubectl get pv

**Step 3:** Create the Persistent volume claim.

* vi pvc.yml

apiVersion: v1

kind: PersistentVolumeClaim

metadata:

  name: my-claim

spec:

  accessModes:

    - ReadWriteOnce

  resources:

    requests:

      storage: 50Gi

* kubectl apply -f pvc.yml
* kubectl get pvc

**Step 4:** Now create a pod which uses the persistent volume “my-claim”.

* vi pod.yml

apiVersion: apps/v1

kind: Deployment

metadata:

  name: pvdeploy

spec:

  replicas: 1

  selector:      *# tells the controller which pods to watch/belong to*

    matchLabels:

     app: mypv

  template:

    metadata:

      labels:

        app: mypv

    spec:

      containers:

      - name: shell

        image: centos

        command: ["bin/bash", "-c", "sleep 10000"]

        volumeMounts:

        - name: mypd

          mountPath: "/tmp/persistent"

      volumes:

        - name: mypd

          persistentVolumeClaim:

            claimName: my-claim

* kubectl apply -f pod.yml
* kubectl get pods

**Liveness probe:** kubernetes only know the pod is running or not, if it is not running it will create automatically.

But, if our pod is running perfectly but our application/service is not working. Then Liveness probe comes into picture, it will check the application health in a predefined interval, if application is running perfectly it will returns 0. If it is not running perfectly then it will return non-zero value. And, create the new pod so that application run perfectly.

* vi livenesprobe.yml

apiVersion: v1

kind: Pod

metadata:

  labels:

    test: liveness

  name: mylivenessprobe

spec:

  containers:

  - name: liveness

    image: ubuntu

    args:

    - /bin/sh

    - -c

    - touch /tmp/healthy; sleep 1000

    livenessProbe:

      exec:

        command:

        - cat

        - /tmp/healthy

      initialDelaySeconds: 5

      periodSeconds: 5

      timeoutSeconds: 30

* kubectl apply -f livenesprobe.yml
* kubectl get pods
* kubectl exec -it mylivenessprobe -- /bin/bash
* cat /tmp/healthy
* echo $?

O/p: 0

* Cat /tmp/arman
* echo $?

O/p: 1

* rm /tmp/healthy
* exit
* kubectl describe pod mylivenessprobe



**Configmap:**

* vi sample.conf
* kubectl create configmap mymap –from-file=sample.conf
* kubectl delete configmap mymap //to delete the configmap
* kubectl get Configmap
* kubectl describe configmap mymap
* vi configmap-pod.yml

apiVersion: v1

kind: Pod

metadata:

  name: myvolconfig

spec:

  containers:

  - name: c1

    image: centos

    volumeMounts:

      - name: testconfigmap

        mountPath: "/tmp/config"   *# the config files will be mounted as Read Only by default here*

  volumes:

  - name: testconfigmap

    configMap:

       name: mymap   *# this should match the config map name created in the first step*

       items:

       - key: sample.conf

         path: sample.conf

* kubectl apply -f configmap-pod.yml
* kubectl get pods
* kubectl exec -it myvolconfig – bin/bash
* cat /tmp/config/sample.conf

O/P: content of sample.conf file.

We have used configmap as volume, now we will see it as environment variable.

* vi configmap-pod-as-env.yml

apiVersion: v1

kind: Pod

metadata:

  name: myenvconfig

spec:

  containers:

  - name: c1

    image: centos

    command: ["/bin/bash", "-c", "while true; do echo Technical-Guftgu; sleep 5 ; done"]

    env:

    - name: MYENV         *# env name in which value of the key is stored*

      valueFrom:

        configMapKeyRef:

          name: mymap      *# name of the config created*

          key: sample.conf

* kubectl apply -f configmap-pod-as-env.yml
* kubectl exec -it myenvconfig -- /bin/bash
* echo $MYENV

O/P: content of sample.conf file.

**Secret:**

* echo "root"> user.txt && echo "mypass123"> password.txt
* ls
* kubectl create secret generic mysecret --from-file=user.txt --from-file=password.txt
* kubectl delete secret mysecret // to delete the secret
* kubectl get secret
* kubectl describe secret mysecret
* vi secret-pod.yml

 apiVersion: v1

kind: Pod

metadata:

  name: myvolsecret

spec:

  containers:

  - name: c1

    image: centos

    command: ["/bin/bash", "-c", "while true; do echo Technical-guftgu; sleep 5 ; done"]

    volumeMounts:

      - name: testsecret

        mountPath: "/tmp/mysecrets"   *# the secret files will be mounted as Read Only by default here*

  volumes:

  - name: testsecret

    secret:

       secretName: mysecret

* kubectl apply -f secret-pod.yml
* kubectl get pods
* kubectl exec -it myvolsecret -- /bin/bash
* cd /tmp/mysecrets

O/P: container can access user.txt password.txt.

**Namespaces:**

* kubectl get namespaces

# How to create own namespaces

* vi dev.yml

apiVersion: v1

kind: Namespace

metadata:

   name: dev

   labels:

     name: dev

* kubectl apply –f dev.yml
* vi pod.yml

kind: Pod

apiVersion: v1

metadata:

  name: testpod

spec:

  containers:

    - name: container01

      image: ubuntu

* kubectl apply -f pod.yml –n dev
* kubectl get pods –n dev
* kubectl config set-context $(kubectl config current-context) --namespace=dev: it will change current namespace to dev namespace.
* kubectl config view | grep namespace: : it will show current set namespace.

**Resource:**

* vi resource-pod.yml

apiVersion: v1

kind: Pod

metadata:

  name: resource-pod-example

spec:

  containers:

  - name: resource

    image: centos

    command: ["/bin/bash", "-c", "while true; do echo Technical-Guftgu; sleep 5 ; done"]

    resources:

      requests:

        memory: "64Mi"

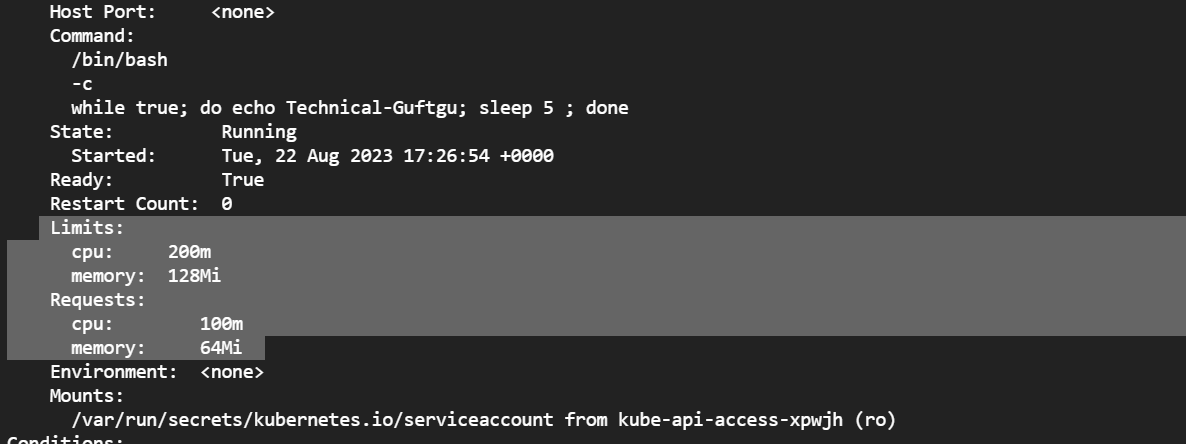
        cpu: "100m"

      limits:

        memory: "128Mi"

        cpu: "200m"

* kubectl apply -f resource-pod.yml
* kubectl describe pod resource-pod-example // we see the detail about the limit and request of resources such as cpu and memory.



**Resource Quota:**

* vi quota.yml

apiVersion: v1

kind: ResourceQuota

metadata:

   name: myquota

spec:

  hard:

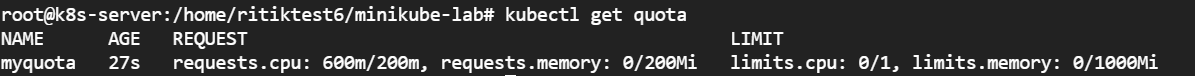
    limits.cpu: "1000m"

    limits.memory: "1000Mi"

    requests.cpu: "200m"

    requests.memory: "200Mi"

* kubectl apply -f quota.yml
* kubectl get quota



* vi use-of-quota-in-pods.yml

kind: Deployment

apiVersion: apps/v1

metadata:

  name: deployments

spec:

  replicas: 3

  selector:

    matchLabels:

     objtype: deployment

  template:

    metadata:

      name: testpod8

      labels:

        objtype: deployment

    spec:

     containers:

       - name: c00

         image: ubuntu

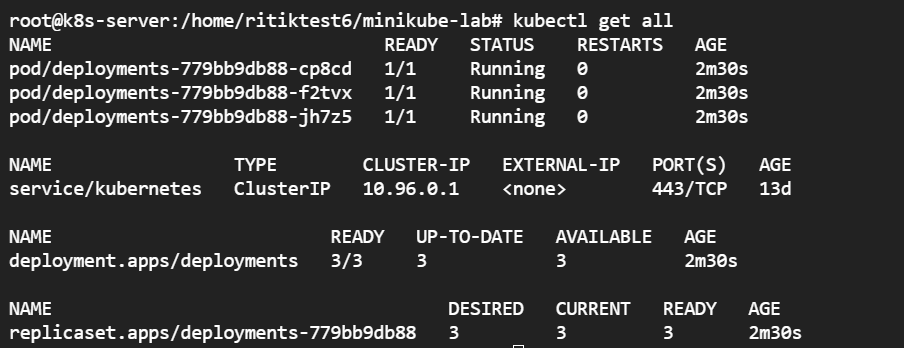
         command: ["/bin/bash", "-c", "while true; do echo Technical-Guftgu; sleep 5 ; done"]

         resources:

            requests:

              cpu: "200m"

* kubectl get all



In quota.yml file we have defined the resources limit, and in use-of-quota-in-pods.yml defined the request of resources, so we have created a deployment in which 3 containers are running with equal resources defined the yml file.

**LimitRange Object:** if we define default request in this yml file. Then each pod created in this cluster default resources will be allocated to all pods.

* vi limitrange.yml

apiVersion: v1

kind: LimitRange

metadata:

  name: cpu-limit-range

spec:

  limits:

  - default:

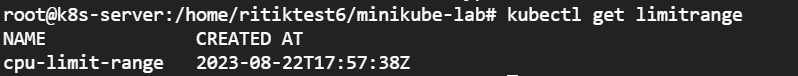
      cpu: 1

    defaultRequest:

      cpu: 0.5

    type: Container

* kubectl apply -f limitrange.yml
* kubectl get limitrange



* vi pod.yml
* Kubectl apply -f pod.yml

kind: Pod

apiVersion: v1

metadata:

 name: testpod

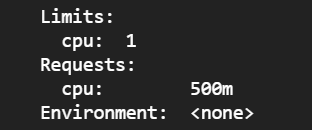
spec:

 containers:

  - name: c00

    image: ubuntu

* kubectl describe pod testpod



**NOTE:**  A container has max and min value of resources.  
(default range) CPU ->   
 (minimum value) request – 0.5  
 (maximum value) limit – 1  
 Memory ->  
 (minimum value) request – 500m  
 (maximum value) limit – 1G

**Auto Scaling:**

* wget -O metricserver.yml <https://github.com/kubernetes-sigs/metrics-server/releases/latest/download/components.yaml>
* vi metricserver.yml and add the given line in this file.
* Add --kubelet-insecure-tls
* kubectl apply -f metricserver.yml
* Kubectl get namespaces -n kube-system
* vi autosacaing.yml

kind: Deployment

apiVersion: apps/v1

metadata:

   name: mydeploy

spec:

   replicas: 1

   selector:

    matchLabels:

     name: deployment

   template:

     metadata:

       name: testpod8

       labels:

         name: deployment

     spec:

      containers:

        - name: c00

          image: httpd

          ports:

          - containerPort: 80

          resources:

            limits:

              cpu: 500m

            requests:

              cpu: 200m

* kubectl apply -f autoscaling.yml
* kubectl autoscale deployment mydeploy --cpu-percent=20 --min=1 --max=10
* kubectl get hpa
* kubectl delete hpa mydeploy

**JOB:** in JOB object, after completing task by container, container terminates itself automatically.

* vi job.yml

apiVersion: batch/v1

kind: Job

metadata:

  name: testjob

spec:

  template:

    metadata:

      name: testjob

    spec:

      containers:

      - name: counter

        image: centos:7

        command: ["bin/bash", "-c", "echo Technical-Guftgu; sleep 5"]

      restartPolicy: Never

* kubectl apply -f job.yml
* kubectl get pods
* kubectl get job
* kubectl delete job <job-name>

**Parallelism in Job:** Parallelism is nothing but a group of containers, which are created to do certain task, and after completing task, all containers deleted.

* vi parallelism.yml

apiVersion: batch/v1

kind: Job

metadata:

  name: testjob

spec:

  parallelism: 5                           *# Runs for pods in parallel*

  activeDeadlineSeconds: 10  *# Timesout after 30 sec*

  template:

    metadata:

      name: testjob

    spec:

      containers:

      - name: counter

        image: centos:7

        command: ["bin/bash", "-c", "echo Technical-Guftgu; sleep 20"]

      restartPolicy: Never

* kubectl apply -f parallelism.yml
* kubectl get job

**Cron Job:** CronJob Object is used to do some task periodically.

* vi cronjob.yml

apiVersion: batch/v1beta1

kind: CronJob

metadata:

 name: mycontainer

spec:

 schedule: "\* \* \* \* \*"

 jobTemplate:

   spec:

     template:

       spec:

         containers:

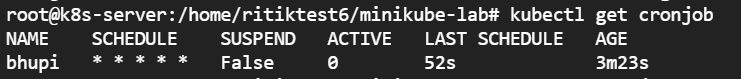
         - image: ubuntu

           name: mycontainer

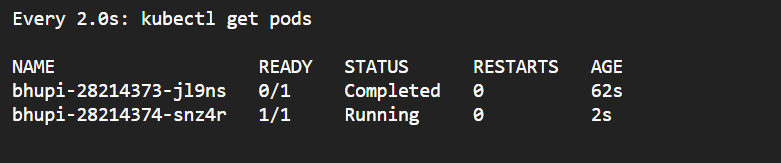
           command: ["/bin/bash", "-c", "echo Technical-Guftgu; sleep 5"]

         restartPolicy: Never

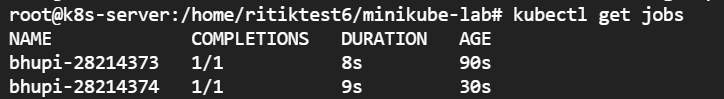
* kubectl apply -f cronjob.yml
* kubectl api-versions | grep batch : to get batch info
* kubectl get cronjob



* watch kubectl get pods



* kubectl get jobs



* kubectl delete cronjob <job-name>
* **Init Container:**  It is like a job, which runs before the actual container.   
  It can be used   
  **Clone a Git repository into a**[**Volume**](https://kubernetes.io/docs/concepts/storage/volumes/)**.**
* vi init-cont.yml

apiVersion: v1

kind: Pod

metadata:

  name: initContainer

spec:

  initcontainers:

  - name: c1

    image: centos

    command: ["/bin/bash", "-c", "echo Arman malik adds by initcontainer > /tmp/xchange; sleep 10"]

    volumeMounts:

      - name: xchange

        mountPath: "/tmp/xchange"

  containers:

  - name: c1

    image: centos

    command: ["/bin/bash", "-c", "sleep 100"]

    volumeMounts:                                    *# Mount definition inside the container*

      - name: xchange

        mountPath: "/tmp/data"

  volumes:

  - name: xchange

    emptyDir: {}

* kubectl apply -f init-cont.yml
* kubectl get pods

**HELM**

* helm repo add [NAME] [URL] [flags]: To add official stable Helm charts.

Eg: helm repo add stable <https://charts.helm.sh/stable> (deprecated)

helm repo add bitnami https://charts.bitnami.com/bitnami

* helm search repo stable: List the contents of the repository.
* helm repo update: update the repo.
* helm repo remove [REPO1 [REPO2 ...]] [flags]: remove repo from the list

Eg: helm repo remove stable.

* helm repo list: to show the list of repo.
* helm search repo <repo-name>/<chart-name>: search the chart in repo.

Eg: helm search repo stable/jenkins.

* helm show values <repo-name>/<chart-name>: to show the values of chart .

Eg: helm show values stable/tomcat.

* helm show values <repo-name>/<chart-name>: It will show the details about chart. Such as vers.

Eg: helm show chart stable/tomcat.

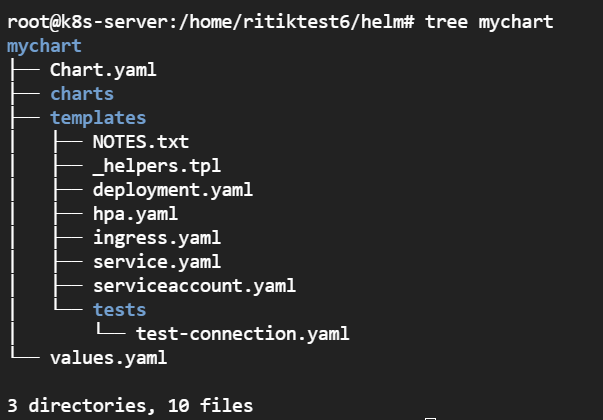
* helm show all <repo-name>/<chart-name>: It will show all details about chart.

Eg: helm show all stable/tomcat.

* helm create <chart-name>: It will create your chart.

Eg: helm create mychart.

tree mychart:



**Understanding the mychart:**

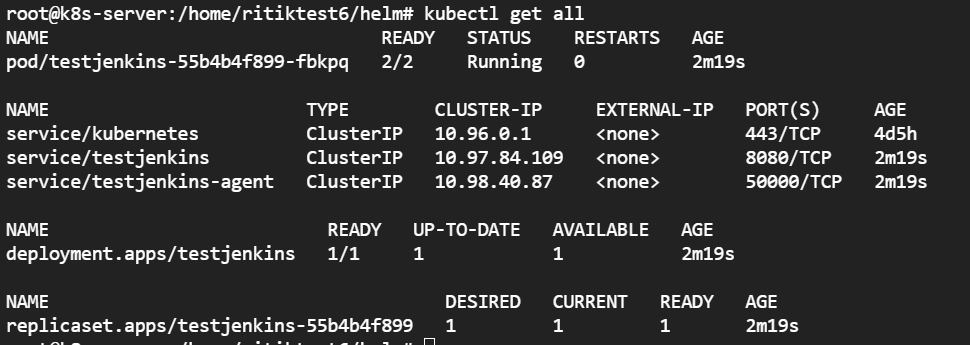
Chart.yaml:

charts:

templates:

values.yaml:

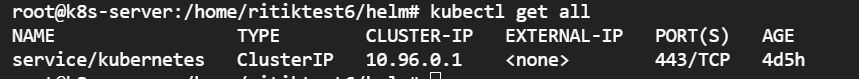
* helm install <release-name> <repo-name>/<chart-name>: it will install chart.
* helm install testjenkins stable/jenkins.
* kubectl get all



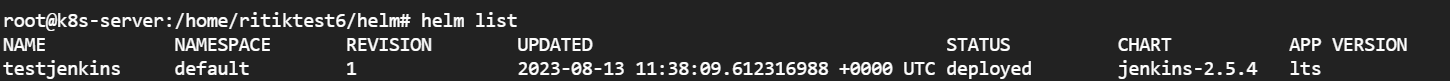
* helm delete <release-name>: It will uninstall the chart

Eg: helm delete testjenkins

* Kubectl get all



* Helm list: It will show the list all the named release.



* helm install –dry-run testchart stable/tomcat: It will just test the chart that will install successfully or get some error. So this command is just used for testing purpose only, it will not create or install anything in actual.
* helm install --wait --timeout 20s testjenkins stable/jenkins: it show the installed result after 20 seconds instead of instant result.
* helm install testjenkins-old stable/jenkins --version 2.5.0: It will installed the specified version of jenkins chart.

**Some Important Commands for helm:** There are two ways to pass configuration data during install.

**- -set:** specify overrides on the command line.

Eg: helm install mychart stable/tomcat --set service.type=NodePort

**- -values (or -f):** specify a YAML file with override

* helm get <all/manifest/values> <release-name>: Information about from a names release.

Eg: helm get all mychart

* helm status <release-name>: Display the status of the named release.

Eg: helm status mychart

* helm history <release-name>: Display the history of the named release.
* helm upgrade <release-name> <chart-name>: Upgrade the release with latest upgraded chart.

Eg: helm upgrade mychart stable/tomcat

* helm rollback <release-name> <Revision>: Rollback a release to any previous version.

Eg: helm rollback mychart: It back to the just previous version.

**Download a chart from a Repository:**

* helm pull <chart-name>: download the tar
* helm pull <chart-name> --untar: untar the chart
* helm pull --untar stable/tomcat: download and untar

**Install from a local chart archive:**

helm install mychart tomcat-0.4.tgz

**Install from a full URL:**

helm install mychart <URL>

HELM HUB: https://artifacthub.io/

**Terraform**

**Hello World Program:**

output "hello" {

  value = "Hello World!"

}

* Terraform plan

Changes to Outputs:

+ hello = "Hello World!"

You can apply this plan to save these new output values to the Terraform state, without changing any real infrastructure.

**Multiple block in a single terraform file:**

output "Name" {

  value = "Arman Malik"

}

output "Age" {

  value = "25"

}

output "City" {

  value = "Noida"

}

* Terraform plan

Changes to Outputs:

+ Age = "25"

+ City = "Noida"

+ Name = "Arman Malik"

You can apply this plan to save these new output values to the Terraform state, without changing any real infrastructure.

**Terraform Variables:**

variable "Name" {

}

output printname {

  value = "Hello ${var.Name}"

}

* Terraform plan

var.Name

Enter a value: Arman

Changes to Outputs:

+ printname = "Hello Arman"

**Use of default keyword:** we don’t pass the value to the variable it will take specified default value.

variable "Name" {

    default = "Malik"

}

output printname {

  value = "Hello ${var.Name}"

}

,

* terraform plan

Changes to Outputs:

+ printname = "Hello Malik"

You can apply this plan to save these new output values to the Terraform state, without changing any real infrastructure.

But if we specified command line argument for the variable it will not print default value.

* terraform plan -var “Name=Arman”

Changes to Outputs:

+ printname = "Hello Arman"

You can apply this plan to save these new output values to the Terraform state, without changing any real infrastructure.

Note: we can pass multiple argument like this.

terraform plan -var “Name=Arman” -var “Age=25”  
  
  
**Data Types in Terraform:**

The Terraform language uses the following types for its values:

* [**string**](https://developer.hashicorp.com/terraform/language/expressions/types#string): a sequence of Unicode characters representing some text, like "hello".
* [**number**](https://developer.hashicorp.com/terraform/language/expressions/types#number): a numeric value. The number type can represent both whole numbers like 15 and fractional values like 6.283185.
* [**bool**](https://developer.hashicorp.com/terraform/language/expressions/types#bool): a boolean value, either true or false. bool values can be used in conditional logic.
* [**list**](https://developer.hashicorp.com/terraform/language/expressions/types#list) (or **tuple**): a sequence of values, like ["us-west-1a", "us-west-1c"]. Elements in a list or tuple are identified by consecutive whole numbers, starting with zero.
* [**map**](https://developer.hashicorp.com/terraform/language/expressions/types#map) (or **object**): a group of values identified by named labels, like {name = "Mabel", age = 52}.

Strings, numbers, and bools are sometimes called *primitive types.* Lists/tuples and maps/objects are sometimes called *complex types,* *structural types,* or *collection types.*

Finally, there is one special value that has *no* type:

* [**null**](https://developer.hashicorp.com/terraform/language/expressions/types#null): a value that represents *absence* or *omission.* If you set an argument of a resource to null, Terraform behaves as though you had completely omitted it — it will use the argument's default value if it has one, or raise an error if the argument is mandatory. null is most useful in conditional expressions, so you can dynamically omit an argument if a condition isn't met.

Example string and number:

variable "name" {

  type = **string**

}

variable "age" {

   type =  **number**

}

output "user" {

    value = "Hello ${var.name} and your age is ${var.age}"

}

* terraform plan

var.age

Enter a value: 25

var.name

Enter a value: Arman

Changes to Outputs:

+ user = "Hello Arman and your age is 25"

Example list:

variable usersName {

  type = **list**

}

output "printIndexedName" {

    value = "Hi ${var.usersName[0]} and ${var.usersName[2]}"

}

* terraform plan

var.usersName

Enter a value: ["Arhan", "Arman", "Arbaz"]

Changes to Outputs:

+ printIndexedName = "Hi Arhan and Arbaz"

**Terraform in-built functions:**

join() / upper() /title()

variable usersName {

  type = **list**

default = ["arhan", "arman", "arbaz"]

}

output "printIndexedName" {

    value = "Hi ${join(",", var.usersName)}"

}

output "UpperName" {

  value = upper(var.usersName[0])

}

output "firstLetterInCaps" {

  value = title(join(" ",var.usersName))

}

output "firstLetterInCaps" {

  value = title(join(" ",var.usersName))

}

* terraform plan

Changes to Outputs:

+ UpperName = "ARHAN"

+ firstLetterInCaps = "Arhan Arman Arbaz"

+ printIndexedName = "Hi Arhan,Arman,Arbaz"

Note: Many built-in functions are present in terraform. Link to know about functions.

<https://developer.hashicorp.com/terraform/language/expressions>

Map variable:

variable "usersAge" {

    type = **map**

    default = {

        Arman = 25

        Shadab = 26

    }

}

variable "userName" {

    type = **string**

}

output "users" {

  value = "Hi my name is ${var.userName} and my age is ${lookup(var.usersAge, "Shadab")}"

}

* terraform plan

var.userName

Enter a value: Shadab­

Changes to Outputs:

+ users = "Hi my name is Shadab and my age is 26"

**Concept of variable.tfvars files:**

Create first.tf file

variable "name" {

  type = **string**

}

variable "age" {

   type =  **number**

}

output "user" {

    value = "Hello ${var.name} and your age is ${var.age}"

}

Create variable.tfvars file with in the same directory.

name = "Arman malik"

age = 25

* terraform apply

Changes to Outputs:

+ user = "Hello Arman malik and your age is 25"

NOTE: variable.tfvars is default variable file but can create custom terraform variables files.

Example:

Create first.tf

variable "name" {

  type = **string**

}

variable "age" {

   type =  **number**

}

output "user" {

    value = "Hello ${var.name} and your age is ${var.age}"

}

developmentVariable.tfvars

name = "Ziyan Malik"

age = 3

* terraform plan -var-file= developmentVariable.tfvars

Error: Too many command line arguments

flag provided but not defined: -chdir

Error: Too many command line arguments

To specify a working directory for the plan, use the global -chdir flag.

**How to read environment variables in tf configuration files:**

variable "userName" {

    type = **string**

}

output "user" {

    value = "Hi ${var.userName}"

}

export TF\_VAR\_userName = 'Sohail' in LINUX

$env:TF\_VAR\_userName = 'Sohail' in PowerShell

* terraform plan

Changes to Outputs:

+ user = "Hi Sohail"

**Understanding of Terraform architecture**

With the help of the preceding section, we learned and became familiar with Terraform, which is just a tool for building, changing, and versioning infrastructure safely and efficiently. Terraform is entirely built on a plugin-based architecture. Terraform plugins enable all developers to extend Terraform usage by writing new plugins or compiling modified versions of existing plugins:

Figure 1.12 – Terraform architecture


**Terraform CORE:**

Terraform Core is a statically compiled binary written in the Go programming language. It uses RPCs to communicate with Terraform plugins and offers multiple ways to discover and load plugins for use. The compiled binary is the Terraform CLI. If you're interested in learning more about this, you should start your journey from the Terraform CLI, which is the only entry point. The code is open source and hosted at [github.com/hashicorp/Terraform](http://github.com/hashicorp/Terraform).

The responsibilities of Terraform Core are as follows:

* IaC: Reading and interpolating configuration files and modules
* Resource state management
* Resource graph construction
* Plan execution
* Communication with plugins via RPC

**Terraform Plugins:**

Terraform plugins are written in the Go programming language and are executable binaries that get invoked by Terraform Core via RPCs. Each plugin exposes an implementation for a specific service, such as AWS, or a provisioner, such as Bash. All providers and provisioners are plugins that are defined in the Terraform configuration file. Both are executed as separate processes and communicate with the main Terraform binary via an RPC interface. Terraform has many built-in provisioners, while providers are added dynamically as and when required. Terraform Core provides a high-level framework that abstracts away the details of plugin discovery and RPC communication, so that developers do not need to manage either.

Terraform plugins are responsible for the domain-specific implementation of their type.

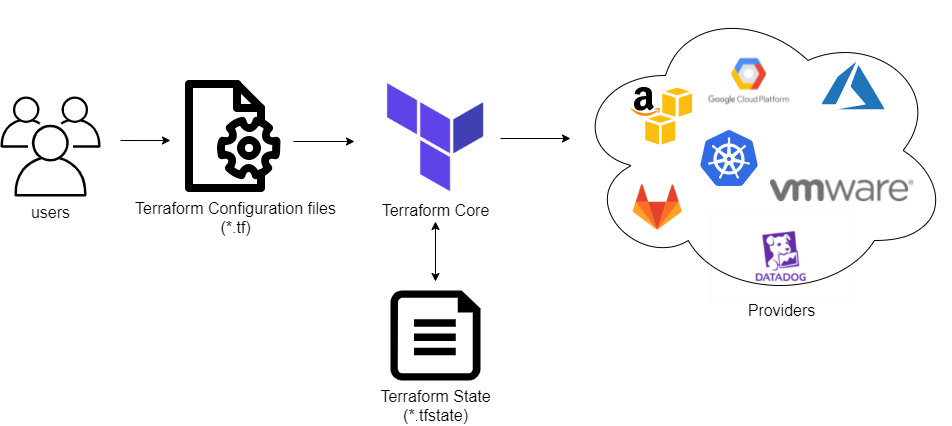
The responsibilities of provider plugins are as follows:

* Initialization of any included libraries used to make API calls
* Authentication with the infrastructure provider
* The definition of resources that map to specific services

The responsibilities of provisioner plugins are as follows:

* Executing commands or scripts on the designated resource following creation or destruction

**How Terraform Works:**



**Main commands:**

init Prepare your working directory for other commands

validate Check whether the configuration is valid

plan Show changes required by the current configuration

apply Create or update infrastructure

destroy Destroy previously-created infrastructure

refresh If you update something on created infra, to update that state in .tfstate file.

show To see the all details of .tfstate file.

**Use of terraform console:**

    variable "userName" {

        type = **string**

        default = "Arman Malik"

    }

    output "user" {

        value = "Hi ${var.userName}"

    }

PS C:\DevOps\Terraform\_LAB\Terraform-Tutorial> terraform console

> var.userName

"Arman Malik"

**Terraform fmt command:**

It is used for apply the good indentation in the terraform configuration files.

TF config file without indentation.

  variable     "userName"  {

        type=     **string**

        default = "Arman Malik"

    }

    output "user" {

        value        ="Hi  ${var.userName}"

    }

* terraform fmt

variable "userName" {

  type    = **string**

  default = "Arman Malik"

}

output "user" {

  value = "Hi  ${var.userName}"

}

terraform taint: not useful command

**Terraform Providers:** A provider in Terraform is a plugin that enables interaction with an API. This includes Cloud providers and Software-as-a-service providers. The providers are specified in the Terraform configuration code. They tell Terraform which services it needs to interact with.

**Example of Terraform Providers:**

* AWS
* Azure
* GCP
* Oracle Cloud
* Alibaba Cloud and so on..

NOTE:

Once a provider is specified, each provider makes a list of resources and data types available for use in the Terraform code. These are listed in the documentation which can be found on the [Terraform Registry.](https://spacelift.io/blog/terraform-registry)

Website link: *https://registry.terraform.io/*

**Terraform Provisioners:** Provisioning mainly deals with configuration activities that happen after the resource is created. It may involve some file operations, executing CLI commands, or even executing the script. Once the resource is successfully initialized, it is ready to accept connections. These connections help Terraform log into the newly created instance and perform these operations.

**Types of Provisioners:**

* Local-exec
* Remote-exec
* File
* Connection: It is required when we are using Remote-exec and File Provisioners.
* Null Resources

**Local-exec Provisioner:** The local-exec provisioner works on the Terraform host – where Terraform configuration is applied/executed. It is used to execute any shell command. It is used to set or read environment variables, details about the resource which is created, invoke any process or application, etc.

**Example:**

provider "google" {

  project     = "dev-ops-project-398806"

  credentials = "${file("credentials.json")}"

  region      = "us-central1"

  zone = "us-central1-c"

}

resource "google\_compute\_instance" "myinstance" {

  name         = "terraform-test1"

  machine\_type = "n1-standard-1"

  zone         = "us-central1-a"

  boot\_disk {

    initialize\_params {

      image = "debian-cloud/debian-11"

      labels = {

        my\_label = "value"

      }

    }

  }

*// Local SSD disk*

  scratch\_disk {

    interface = "SCSI"

  }

  network\_interface {

    network = "default"

  }

  desired\_status = "RUNNING"

 allow\_stopping\_for\_update=true

 provisioner "local-exec" {

*#command = "echo ${self.network\_ip} >> network\_ip.txt"*

   command = "echo I created the resource.."

 }

}

**NOTE:** The “when” attribute used in the provisioner block determines whether a provisioner is creation-time or destroy-time.

provider "google" {

  project     = "dev-ops-project-398806"

  credentials = "${file("credentials.json")}"

  region      = "us-central1"

  zone = "us-central1-c"

}

resource "google\_compute\_instance" "myinstance" {

  name         = "terraform-test2"

  machine\_type = "n1-standard-1"

  zone         = "us-central1-a"

  boot\_disk {

    initialize\_params {

      image = "debian-cloud/debian-11"

      labels = {

        my\_label = "value"

      }

    }

  }

*// Local SSD disk*

  scratch\_disk {

    interface = "SCSI"

  }

  network\_interface {

    network = "default"

  }

  desired\_status = "RUNNING"

 allow\_stopping\_for\_update=true

 provisioner "local-exec" {

  when = destroy

   command = "echo I Have Created.."

 }

provisioner "local-exec" {

  when = destroy

   command = "echo I Have Destroyed.."

 }

}

**Remote-exec:** The remote-exec provisioners are similar to local-exec provisioners – where the commands are executed on the target EC2 instance instead of Terraform host.

**Example:**

resource "aws\_instance" "my\_vm" {

ami = var.ami //Amazon Linux AMI

instance\_type = var.instance\_type

key\_name = "tfsn"

security\_groups = [aws\_security\_group.http\_access.name]

provisioner "remote-exec" {

inline = [

"touch hello.txt",

"echo 'Have a great day!' >> hello.txt"

]

}

connection {

type = "ssh"

host = self.public\_ip

user = "ec2-user"

private\_key = file("./tfsn.cer")

timeout = "4m"

}

tags = {

Name = var.name\_tag,

}

}

**File Provisioning:** The file provisioner is a way to copy certain files or artifacts from the host machine to target resources that will be created in the future. This is a very handy way to transport certain script files, configuration files, artifacts like .jar files, binaries, etc. when the target resource is created and boots for the first time.

**Example:**

resource "aws\_instance" "my\_vm" {

ami = var.ami //Amazon Linux AMI

instance\_type = var.instance\_type

key\_name = "tfsn"

security\_groups = [aws\_security\_group.http\_access.name]

provisioner "file" {

source = "./letsdotech.txt"

destination = "/home/ec2-user/letsdotech.txt"

}

connection {

type = "ssh"

host = self.public\_ip

user = "ec2-user"

private\_key = file("./tfsn.cer")

timeout = "4m"

}

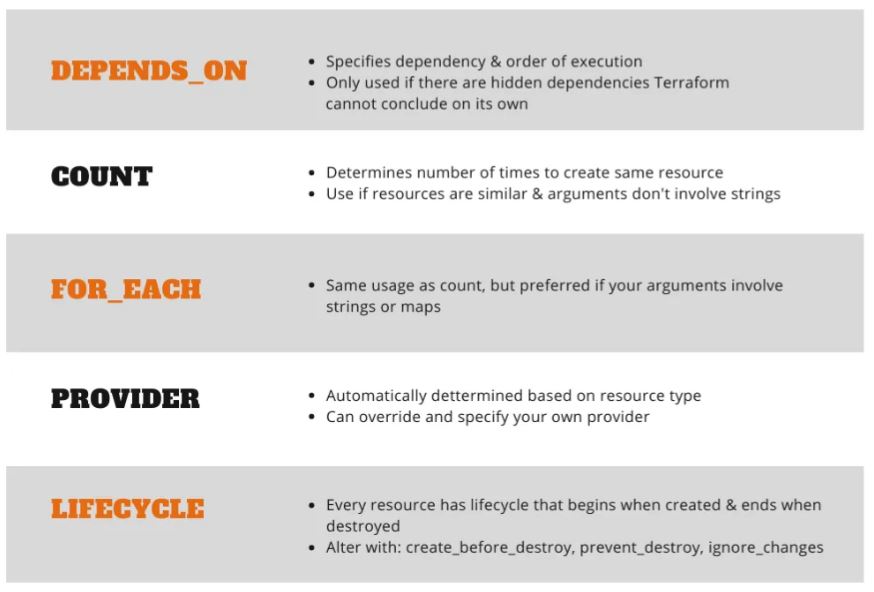
tags = {

Name = var.name\_tag,

}

}

**Resource Meta-Arguments:**



**DEPENDS\_ON:**

provider "google" {

  project     = "dev-ops-project-398806"

  credentials = "${file("credentials.json")}"

  region      = "us-central1"

  zone = "us-central1-c"

}

resource "google\_compute\_instance" "myinstance" {

  name         = "terraform-test"

  machine\_type = "n1-standard-1"

  zone         = "us-central1-a"

  boot\_disk {

    initialize\_params {

      image = "debian-cloud/debian-11"

      labels = {

        my\_label = "value"

      }

    }

  }

*// Local SSD disk*

  scratch\_disk {

    interface = "SCSI"

  }

   network\_interface {

    network = "default"

      access\_config {

    }

  }

  desired\_status = "RUNNING"

  allow\_stopping\_for\_update=true

  depends\_on = [google\_storage\_bucket.static-site]

}

resource "google\_storage\_bucket" "static-site" {

  name          = "arman-demo-buck07891"

  location      = "EU"

  force\_destroy = true

  uniform\_bucket\_level\_access = true

  cors {

    origin          = ["http://arman-demo-buck0789"]

    method          = ["GET", "HEAD", "PUT", "POST", "DELETE"]

    response\_header = ["\*"]

    max\_age\_seconds = 3600

  }

}

**COUNT:**

provider "google" {

  project     = "dev-ops-project-398806"

  credentials = "${file("credentials.json")}"

  region      = "us-central1"

  zone = "us-central1-c"

}

resource "google\_compute\_instance" "myinstance" {

  count        = 3

  name         = "terraform-${count.index}"

  machine\_type = "n1-standard-1"

  zone         = "us-central1-a"

  boot\_disk {

    initialize\_params {

      image = "debian-cloud/debian-11"

      labels = {

        my\_label = "label1"

      }

    }

  }

*// Local SSD disk*

  scratch\_disk {

    interface = "SCSI"

  }

   network\_interface {

    network = "default"

      access\_config {

    }

  }

  desired\_status = "RUNNING"

  allow\_stopping\_for\_update=true

}

**FOR\_EACH:**

Ex1: Map

provider "google" {

  project     = "dev-ops-project-398806"

  credentials = "${file("credentials.json")}"

  region      = "us-central1"

  zone = "us-central1-c"

}

locals {

  compute\_names = {developer="us-central1-a", tester="us-central1-b"}

}

resource "google\_compute\_instance" "myinstance" {

  for\_each     = tomap(local.compute\_names)

  name         = each.key

  machine\_type = "n1-standard-1"

  zone         = each.value

  boot\_disk {

    initialize\_params {

      image = "debian-cloud/debian-11"

      labels = {

        my\_label = "label1"

      }

    }

  }

*// Local SSD disk*

  scratch\_disk {

    interface = "SCSI"

  }

   network\_interface {

    network = "default"

      access\_config {

    }

  }

  desired\_status = "RUNNING"

  allow\_stopping\_for\_update=true

}

Ex2: Set of strings:

provider "google" {

  project     = "dev-ops-project-398806"

  credentials = "${file("credentials.json")}"

  region      = "us-central1"

  zone = "us-central1-c"

}

locals {

  compute\_names = ["dev-1", "dev-2"]

}

resource "google\_compute\_instance" "myinstance" {

  for\_each     = toset(local.compute\_names) *#it will create the 2 instances dev-1 and dev-2*

  name         = each.key

  machine\_type = "n1-standard-1"

  zone         = "us-central1-a"

  boot\_disk {

    initialize\_params {

      image = "debian-cloud/debian-11"

      labels = {

        my\_label = "label1"

      }

    }

  }

*// Local SSD disk*

  scratch\_disk {

    interface = "SCSI"

  }

   network\_interface {

    network = "default"

      access\_config {

    }

  }

  desired\_status = "RUNNING"

  allow\_stopping\_for\_update=true

}

**LifeCycle: create\_before\_destroy**

This tells Terraform to create a new compute instance with the updated configuration before destroying the old compute instance.

provider "google" {

  project     = "dev-ops-project-398806"

  credentials = "${file("credentials.json")}"

  region      = "us-central1"

  zone = "us-central1-c"

}

resource "google\_compute\_instance" "myinstance" {

  name         = "my-instance"

  machine\_type = "n1-standard-1"

  zone         = "us-central1-a"

  boot\_disk {

    initialize\_params {

      image = "debian-cloud/debian-11"

      labels = {

        my\_label = "label1"

      }

    }

  }

*// Local SSD disk*

  scratch\_disk {

    interface = "SCSI"

  }

   network\_interface {

    network = "default"

      access\_config {

    }

  }

  desired\_status = "RUNNING"

  allow\_stopping\_for\_update=true

   lifecycle {

    create\_before\_destroy = true

  }

}

**Prevent\_destroy:**

In this example, the **prevent\_destroy**argument is set to true, which prevents the EC2 instance from being destroyed. This is useful when you want to make sure that certain resources are not accidentally deleted, such as production databases or key management systems or critical production machines, etc.

**ignore\_changes**

The **ignore\_changes**feature in Terraform is useful in situations where a resource is created with references to data that may change in the future, but those changes should not affect the resource after its creation.

**Ansible**

**Installation and Setting up Ansible on Root and Nodes**

$ sudo apt update

$ sudo apt install software-properties-common arman

$ sudo add-apt-repository --yes --update ppa:ansible/ansible

$ sudo apt install ansible

After installation

Open the hosts file and create the group and add internal IPs of the nodes.

vi /etc/ansible/hosts

[demo] #group name

10.182.0.3

10.182.0.4

Save and exit

vi /etc/ansible/ansible.cfg

uncomment the “inventory” and “sudo\_user” line

*#inventory      = /etc/ansible/hosts*

*#library        = /usr/share/my\_modules/*

*#module\_utils   = /usr/share/my\_module\_utils/*

*#remote\_tmp     = ~/.ansible/tmp*

*#local\_tmp      = ~/.ansible/tmp*

*#plugin\_filters\_cfg = /etc/ansible/plugin\_filters.yml*

*#forks          = 5*

*#poll\_interval  = 15*

*#sudo\_user      = root*

*#ask\_sudo\_pass = True*

*#ask\_pass      = True*

*#transport      = smart*

*#remote\_port    = 22*

*#module\_lang    = C*

*#module\_set\_locale = False*

Adduser ansible #do this on Master and all nodes.

su – ansible #do this on Master and all nodes.

Exit as ansible user

#do this on Master and all nodes.

Give the previllege to the ansible user: visudo

And add the given line

ansible ALL=(ALL:ALL) NOPASSWD: ALL

#do this on Master and all nodes.

vi /etc/ssh/sshd\_config

# Authentication:

#LoginGraceTime 2m

PermitRootLogin prohibit-password #uncomment krna h

#StrictModes yes

#MaxAuthTries 6

#MaxSessions 10

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

PasswordAuthentication yes #uncomment krna h

#PermitEmptyPasswords no

Exit

#do this on Master and all nodes.

service sshd restart

Now create the TRUST relationship

At Master node as ansible user

ssh-keygen

ssh-copy-id [ansible@10.182.0.3](mailto:ansible@10.182.0.3)

enter passwd

ssh-copy-id [ansible@10.182.0.4](mailto:ansible@10.182.0.4)

enter psasswd

Now we do not need to enter passwd again and again.

**HOST PATTERNS:**

all patterns refer to all the machines in the inventory.

ansible all --list-hosts

ansible <groupname> --list-hosts

ansible <groupname>[0] --list-hosts

NOTE:

<groupname>[0] refers to 1st machine/node of the group.

<groupname>[1] refers to 2nd machine/node of the group.

<groupname>[-1] refers to last machine/node of the group.

<groupname>[1:3] refers from 2nd to 4th machine/node of the group.

Group separated by colon can be used to use hosts from multiple group.

groupname1: groupname2

ex. Demo[0:3]:Developer[2:4]

**Ansible working:** there are 3 ways to use ansible as configuration tool

* 1. Ad-hoc commands : simple linux commands
  2. Module : single task
  3. Playbooks : more than one module (multiple module)

1. **Ad-hoc commands in ansible:**  ad-hoc means temporary. Ad-hoc commands are just normal Linux commands.

**Note:**

There is no Idempotency in Ad-hoc commands.

It means if you create a file named as “myfile” and after sometime, you create a file as “myfile”. Ad-hoc commands just overwrite the file. if you do so again and again. It will create file all time and overwrite it.

In short ad-hoc commands just do the task again and again even task is done already.

These ad-hoc commands are not used for configuration management and deployment, because these commands are one time usage.

The ansible ad-hoc commands uses /usr/bin/ansible command line tool to automate a single task.

**Some ansible ad-hoc commands**

ansible demo -a “ls” #**demo** is group name and **-a** **“ls”** is the argument.

ansible demo[0] -a “touch file1”

ansible all -a “mkdir mydir”

ansible demo -a “sudo apt install apache2 -y” OR ansible demo -ba “apt install apache2 -y"

NOTE: in ansible -b is treated as sudo.

ansible demo -ba “apt remove apache2 -y"

**Ansible Module:** ansible ships with a number of modules (called ‘module library’) that can be executed directly on remote hosts or through ‘playbooks’.

Your library of modules can reside on any machine, and there are no servers daemons, or database required.

**Q. where ansible modules are stored?**

**Ans.** In the default location for the inventory file is /etc/ansible/hosts.

**Ansible module commands:** -m means module/-b means sudo privileges

ansible demo -b -m apt -a “pkg=apache2 state=present” #present means install

ansible demo -b -m apt -a “pkg=apache2 state=latest” #latest means update

ansible demo -b -m apt -a “pkg=apache2 state=absent” #absent means remove

ansible demo -b -m service -a “pkg=apache2 state=started”

ansible demo -b -m user -a “name=arman state=absent” #deleting user

ansible demo -b -m copy -a “src=myfile dest=/tmp” #copying file from master to group demo machines.

ansible demo -m setup #give the current state info of the all machine present in group demo

ansible demo[0] -m setup #give the current state info of the 1st machine present in group demo

ansible demo -m setup -a “filter=\*ipv4\*”

**Ansible Playbooks:**

Playbook in ansible are written in YAML format.

It is commonly used for configuration files.

Playbook is like a file where you write codes consists of vars, tasks, handlers, templates and roles.

Each playbook is composed of one or more modules in a list.

Playbook are divided into many sections:

Target Section: Define the host against which playbooks task has to be executed.

Variable Section: Define variables .

Task Section: List of all modules that need to run in an order.

**Ansible Playbook Lab:**

**LAB 1:**

vi target.yml

--- *# My first ansible playbook (target demo)*

- hosts: demo

user: ansible

become: yes         *# It means giving sudo privilege to the ansible user.*

connection: ssh

gather\_facts: yes   *# It provides the nodes information (give Private IP) of demo group.*

ansible-playbook target.yml

**LAB 2:**

vi tasks.yml

--- *# Target and Task ansible playbook*

*#target part*

- hosts: demo

  user: ansible

  become: yes

  connection: ssh

*#Task part*

  tasks:

    - name: Installation of apache2 server *#This is just a msg to show on console what task is going on.*

      action: apt name=apache2 state=present

ansible-playbook tasks.yml

**LAB 3:**

vi variable.yml

--- *# Target and Task ansible playbook*

*#target part*

- hosts: demo

  user: ansible

  become: yes

  connection: ssh

  vars:

    packagename: tree

*#Task part*

  tasks:

    - name: Installation of tree package.

      action: apt name='{{packagename}}' state=present

ansible-playbook variable.yml

**Handler section:** A handler is exactly the same as task, but it will run when called by another task.

Or

Handlers are just like regular tasks in an ansible playbook, but are only run if the task contains a **notify** directive and also indicates that it changed something.

**DRY RUN**

**Special note:** check whether the playbook is formatted correctly.

ansible-playbook handlers.yml --check

vi handlers.yml

--- *# Target and Task anible playbook*

*#target part*

- hosts: demo

  user: ansible

  become: yes

  connection: ssh

  vars:

    packagename: apache2

*#Task part*

  tasks:

    - name: Installation of tree package.

      action: apt name='{{packagename}}' state=present

      notify: restart apache2

  handlers:

    - name: restart apache2

      action: service name='{{packagename}}' state=restarted

ansible-playbook handlers.yml

**Loops in ansible playbook:**

vi loop.yml

--- *# Target and Task ansible playbook*

*#target part*

- hosts: demo

  user: ansible

  become: yes

  connection: ssh

*#Task part*

  tasks:

    - name: Adding users

      user: name='{{item}}' state=present

      with\_items:

        - Arman

        - Arhan

        - Huzefa

ansible-playbook loop.yml

To verify go inside the node1 and run *cat /etc/passwd*

**Conditions in ansible playbook:**  *“when”* statement is used for applying conditions.

And also sometime you want to skip some operation on a particular node.

vi condition.yml

--- *# Target and Task ansible playbook*

*#target part*

- hosts: demo

  user: ansible

  become: yes

  connection: ssh

*#Task part*

  tasks:

    - name: Installation of apache2 package on Debian machines.

      command: apt install apache2 -y

      when: ansible\_os\_family == "Debian"

    - name: Installation of httpd package on RedHat machines.

      command: yum install httpd -y

      when: ansible\_os\_family == "RedHat"

ansible-playbook condition.yml

**Vault:** Ansible allows keeping sensitive data such as passwords or keys in encrypted files, rather than a plain-text 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 existing playbook

Ansible-vault decrypt target.yml

**Roles in ansible:**  We can use two techniques for reusing a set of tasks:- includes and roles.

Roles are good for organizing tasks and encapsulating data needed to accomplish those tasks.

playbook

roles

Master.yml

**Ansible Roles:**

handlers

vars

tasks

main.yml

main.yml

main.yml

myrole

target

roles:

myrole

* 1. default
  2. files
  3. handlers
  4. meta
  5. templates
  6. tasks
  7. vars
* we can organize playbooks into a directory structure called roles.
* Adding more and more functionality to the playbooks will make it difficult to maintain in a single playbook file.

**Hands on ansible roles**

* mkdir -p playbook/roles/webserver/tasks
* tree
* cd playbook
* touch master.yml
* touch roles/webserver/tasks/main.yml
* vi roles/webserver/tasks/main.yml

    - name: Installation of apache2 server*.*

      action: apt name=apache2 state=present

* vi master.yml

- hosts: demo

  user: ansible

  become: yes

  connection: ssh

  roles:

    - webserver

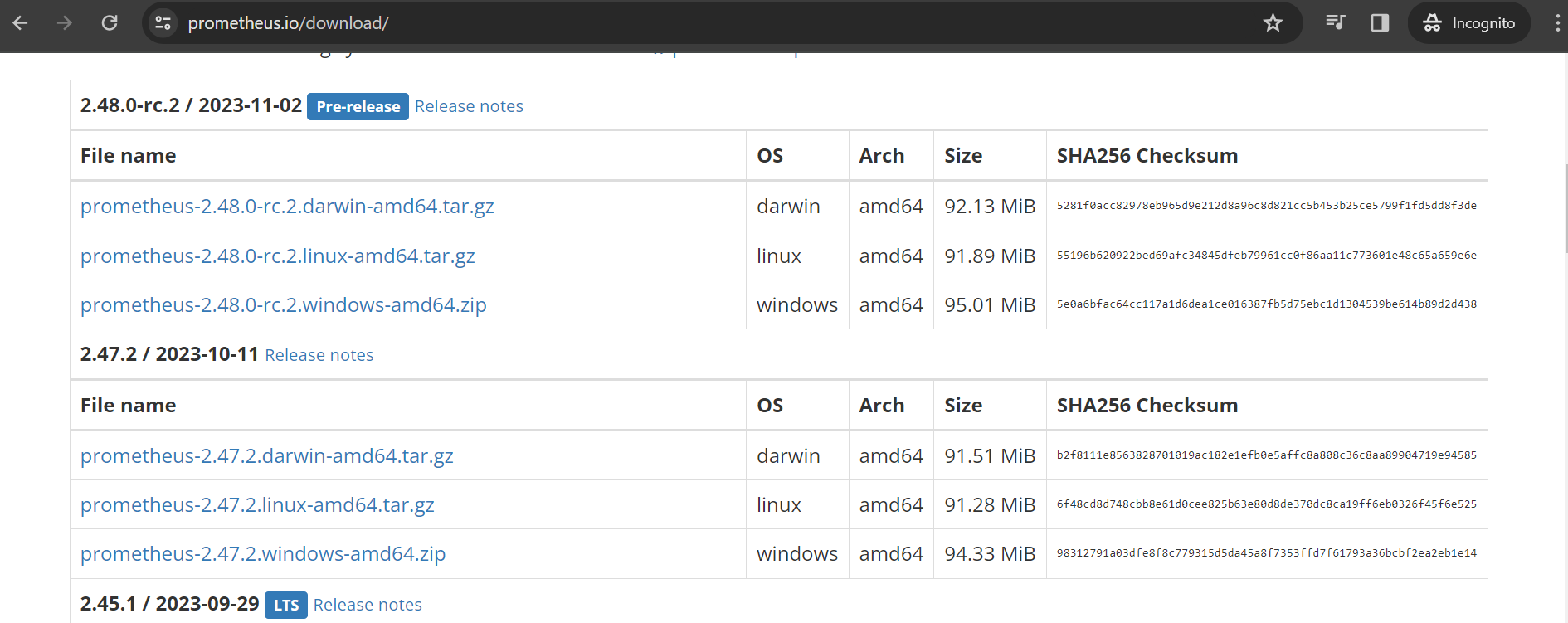
ansible-playbook master.yml

**Prometheus**

**Download and Installation of Prometheus:**

**STEP 1:**

Go to official site of Prometheus and download the Prometheus package.



And run the following command

wget <copied URL>

**STEP 2:**

Unzip the file using following command

tar zxvf <Prometheus-file.tar>

**STEP 3:**

Go to Prometheus folder

cd prometheus-2.48.0-rc.2.linux-amd64

**STEP 4:**

Starting Prometheus

./prometheus --config.file=prometheus.yml

**STEP 5:**

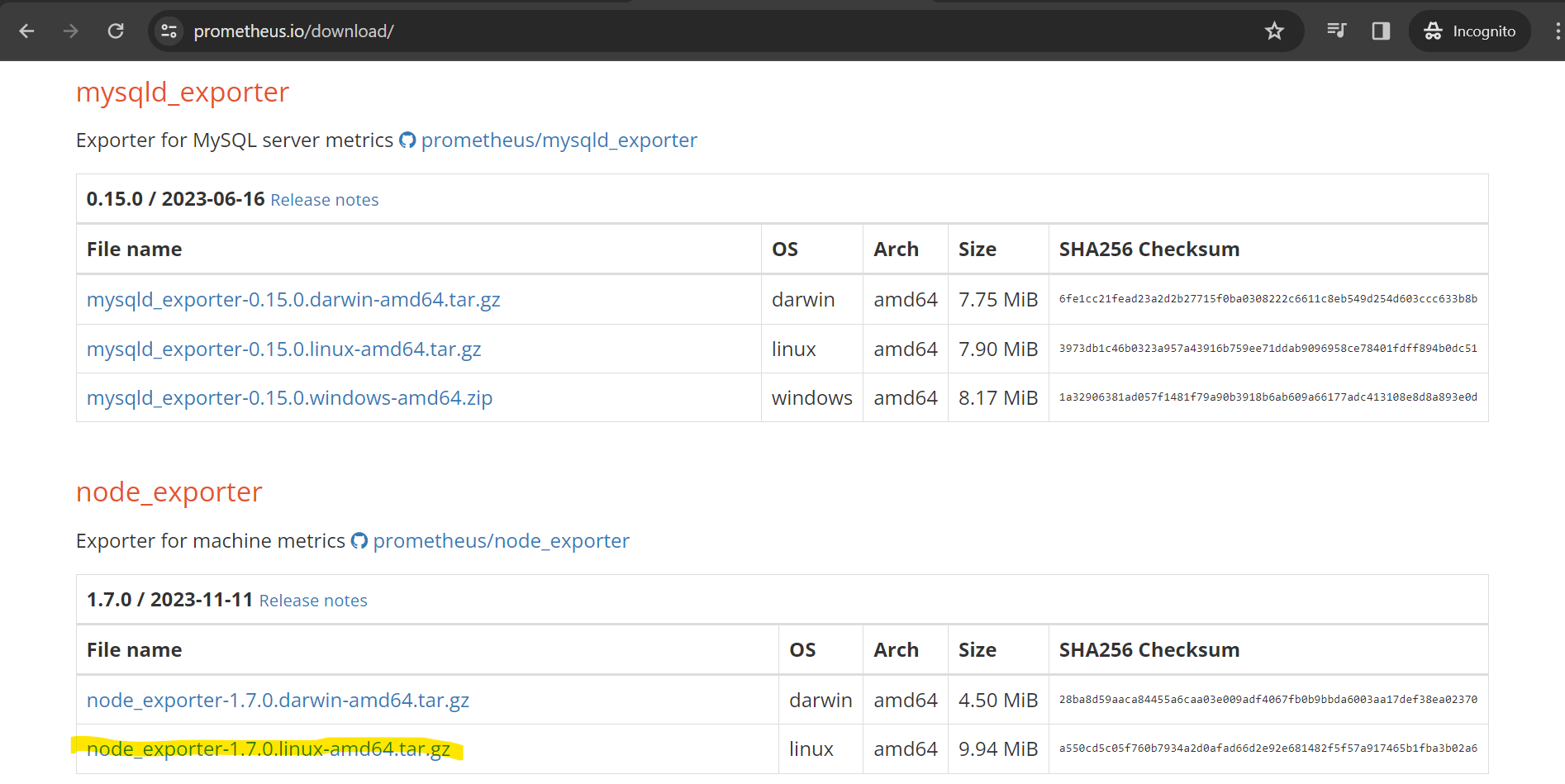
Access the Prometheus

<Public\_IP:9090>

**Download and Installation of Node Exporter in Prometheus:**

**STEP 1:**

Go to official site of Prometheus and download the Node Exporter on different machine (ex node-1).



And run the following command

wget <copied URL>

**STEP 2:**

Unzip the file using following command

tar zxvf <node-exporter.tar>

**STEP 3:**

Go to node-exporter folder

cd node\_exporter-1.7.0.linux-amd64

**STEP 4:**

Starting Node\_exporter

./node\_exporter

**STEP 5:**

Access the Node\_exporter

<Public\_IP:9100>

**Grafana**

**CSS COLORS:** https://palettes.shecodes.io/

**Download and Installation of Prometheus:** Follow the steps given in this link <https://grafana.com/docs/grafana/latest/setup-grafana/installation/debian/>

**Starting a Grafana Server:**

* sudo systemctl daemon-reload
* sudo systemctl start grafana-server
* sudo systemctl status grafana-server

**If Service is not enable then 🡪**

* sudo systemctl enable grafana-server.service
* sudo service grafana-server restart

**NOTE:**  The default port number of Grafana-server is 3000.

i.e. <External-IP>:3000