

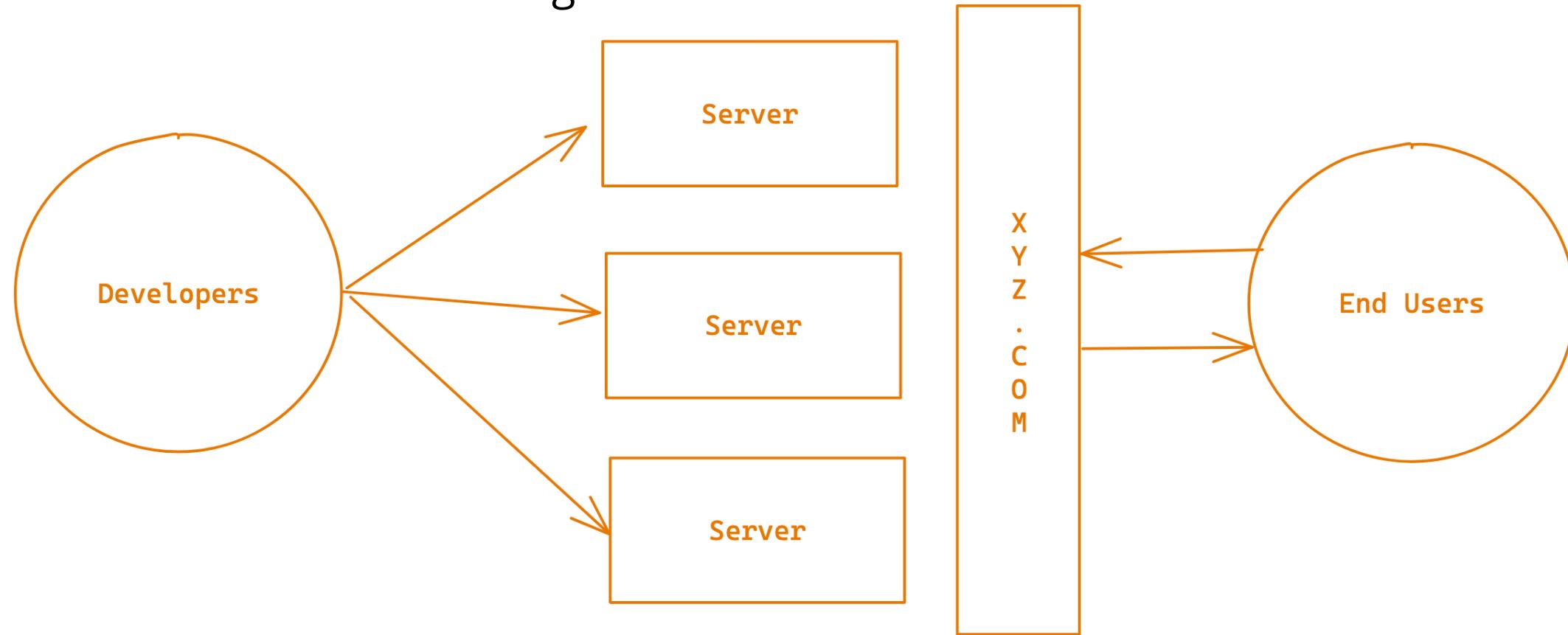
Roles of a DevOps Engineer

- Take code from Developers and deploy the code to multiple environments with minimal downtime.
- Ensure we are keeping the application uptime to 99.9%

What kind of Apps?

Web Apps

Role1: Ensure the code from Developer taken to client/customer/enduser whom ever is accessing



Role2: Ensure these env available all the time with 99.99% uptime

Which Servers & Server OS

Linux

Which Linux

Which Linux Vendor

- RedHat
- Ubuntu
- SUSE

RHEL(RedHat Enterprise Linux)

- Not Free
- But it is OpenSource
- Code can be modified & Can be re-distributed.

Who redistributes

- **CentOS**
- Oracle
- Amazon

Which CentOS

- 7
- 8 (Actively Highly Used one)

Where to run Linux?

- A local Virtual machine in Desktop
- Create a server in AWS or any cloud.
- **Using Platform like Killercoda**

How we run our sessions?

Create a AWS Account

Linux Basics

File Commands

ls

touch

rm

cp

mv

1. Everything is a command.
2. Commands is what we in linux sessions
3. From command line you can find certain info
 - a. username
 - b. hostname
 - c. \$ - normal user, # - root user
root - administrator in Linux
4. Commands will have options
 - single-word
 - single-word
 - s (single character)

1. Machines in todays world we use is not physical directly those are virtual machines

1. To list the files you have ls

2. output of ls command will show files and directories, Directories can be identified with some blue color, also ls -l command as well

3. You can combine options -lA , -Al , However this combination is purely depends on the command.. Meaning every command in linux may not be flexible with combining options. Thus my suggestion is use individual.

1. Linux has case-sensitive file systems. Meaning in Windows if you create a file ABC.txt, you cannot create abc.txt. So in Linux you can create ABC.txt and abc.txt and also Abc.txt also.
2. Windows works on extention like .txt, .mp3, .doc. But Linux does not need any extention..

In windows file abc.txt denotes

- > abc is file name
- > .txt is extention of the file.

In Linux abc.txt denotes

- > abc.txt is filename

The extention of files are created in Linux only for our understanding that what type of file it is

.py -> pythonfile

.sh -> shell file

However we will prefer to use extentions to make our life easy.

Directory Commands

pwd

cd

mkdir

rmdir

rm -r

cp -r

mv

File Content Commands

cat

head

tail

grep

awk

vim

Utility Commands

find

curl

tar

unzip

Pipes (|)

AWS Account

Choose Region

AWS Management Console

Choose a AWS Region

Choosing N.Virginia, Because of pricing and availability

Stay connected to your go

Download the AWS Co Android mobile device

Explore AWS

Free Digital Training

Amazon SageMaker Autopilot

RDS Read Replicas

AWS Certification

US East (N. Virginia) us-east-1

US East (Ohio) us-east-2

US West (N. California) us-west-1

US West (Oregon) us-west-2

Africa (Cape Town) af-south-1

Asia Pacific (Hong Kong) ap-east-1

Asia Pacific (Mumbai) ap-south-1

Asia Pacific (Seoul) ap-northeast-2

Asia Pacific (Singapore) ap-southeast-1

Asia Pacific (Sydney) ap-southeast-2

Asia Pacific (Tokyo) ap-northeast-1

Canada (Central) ca-central-1

Europe (Frankfurt) eu-central-1

Europe (Ireland) eu-west-1

Europe (London) eu-west-2

Europe (Milan) eu-south-1

Europe (Paris) eu-west-3

Europe (Stockholm) eu-north-1

Middle East (Bahrain) me-south-1

South America (São Paulo) sa-east-1

The screenshot shows the AWS Management Console homepage. On the right side, there is a prominent 'Choose a AWS Region' dialog box. Inside the dialog, a list of AWS regions is displayed, with 'N. Virginia' highlighted by a pink arrow. A black arrow points from the text 'Choosing N.Virginia, Because of pricing and availability' to the same 'N. Virginia' option. The dialog also contains other text like 'Stay connected to your go' and 'Download the AWS Co Android mobile device'. Below the dialog, the main AWS services navigation bar is visible, showing sections for Compute, Developer Tools, Machine Learning, Front-end Web & Mobile, AR & VR, Application Integration, and more. The 'Recently visited services' section is also present.

Availability Zones

Service health	
	Service Health Dashboard
Region	Status
US East (N. Virginia)	
Zone status	
Zone	Status
us-east-1a (use1-az2)	Zone is operating normally
us-east-1b (use1-az4)	Zone is operating normally
us-east-1c (use1-az6)	Zone is operating normally
us-east-1d (use1-az1)	Zone is operating normally
us-east-1e (use1-az3)	Zone is operating normally
us-east-1f (use1-az5)	Zone is operating normally
Enable additional Zones	

AZ is nothing but one data center

Services

AWS Management Console

SERVICES ARE GROUPED

The screenshot shows the AWS Management Console's 'Services' page. At the top left is a search bar with placeholder text 'Example: Relational Database Service, database, RDS'. Below it is a section for 'Recently visited services'. The main area is titled 'All services' and contains several groups of services:

- Compute**: EC2, Lightsail, Lambda, Batch, Elastic Beanstalk, Serverless Application Repository, AWS Outposts, EC2 Image Builder.
- Containers**: Elastic Container Registry, Elastic Container Service, Elastic Kubernetes Service.
- Storage**: S3, EFS.
- Developer Tools**: CodeStar, CodeCommit, CodeArtifact, CodeBuild, CodeDeploy, CodePipeline, Cloud9, X-Ray.
- Customer Enablement**: AWS IQ, Support, Managed Services.
- Robotics**: AWS RoboMaker.
- Machine Learning**: Amazon SageMaker, Amazon Augmented AI, Amazon CodeGuru, Amazon Comprehend, Amazon Forecast, Amazon Fraud Detector, Amazon Kendra, Amazon Lex, Amazon Personalize, Amazon Polly, Amazon Rekognition, Amazon Textract, Amazon Transcribe, Amazon Translate.
- Front-end Web & Mobile**: AWS Amplify, Mobile Hub, AWS AppSync, Device Farm.
- AR & VR**: Amazon Sumerian.
- Application Integration**: Step Functions, Amazon AppFlow, Amazon EventBridge, Amazon MQ, Simple Notification Service, Simple Queue Service, SWF.
- Customer Engagement**.

On the right side of the page, there are sections for 'Stay connected to your AWS resources on-the-go' (with a link to the AWS Console Mobile App), 'Explore AWS' (with links to Free Digital Training, Amazon SageMaker Autopilot, RDS Read Replicas, and AWS Certification), and 'Have feedback?'.

Servers Service is EC2

- Servers are Virtual and Physical
- Physical Servers are referred to as **metal**.
- Most of the time it is Virtual Machine

Server

1. VMWare
2. XEN
3. KVM



Desktop

1. VMWare Workstation
2. Oracle VBox
3. Parallels

Admin Commands

ps

kill

groupadd

useradd

usermod

visudo

sudo

su

yum install

yum remove

yum update

chmod

chgrp

chown

ip a

netstat -lntp

telnet

systemctl enable

systemctl start

systemctl stop

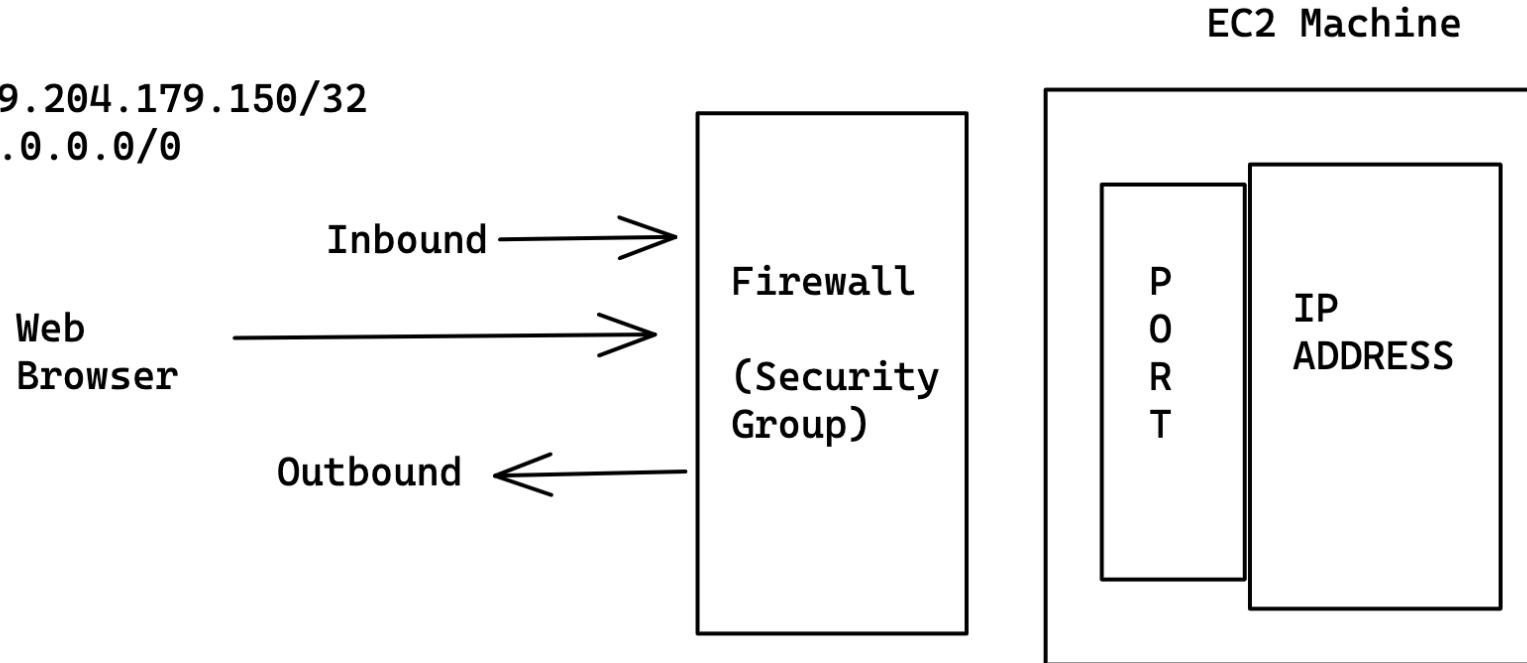
systemctl restart

Firewall

Source:

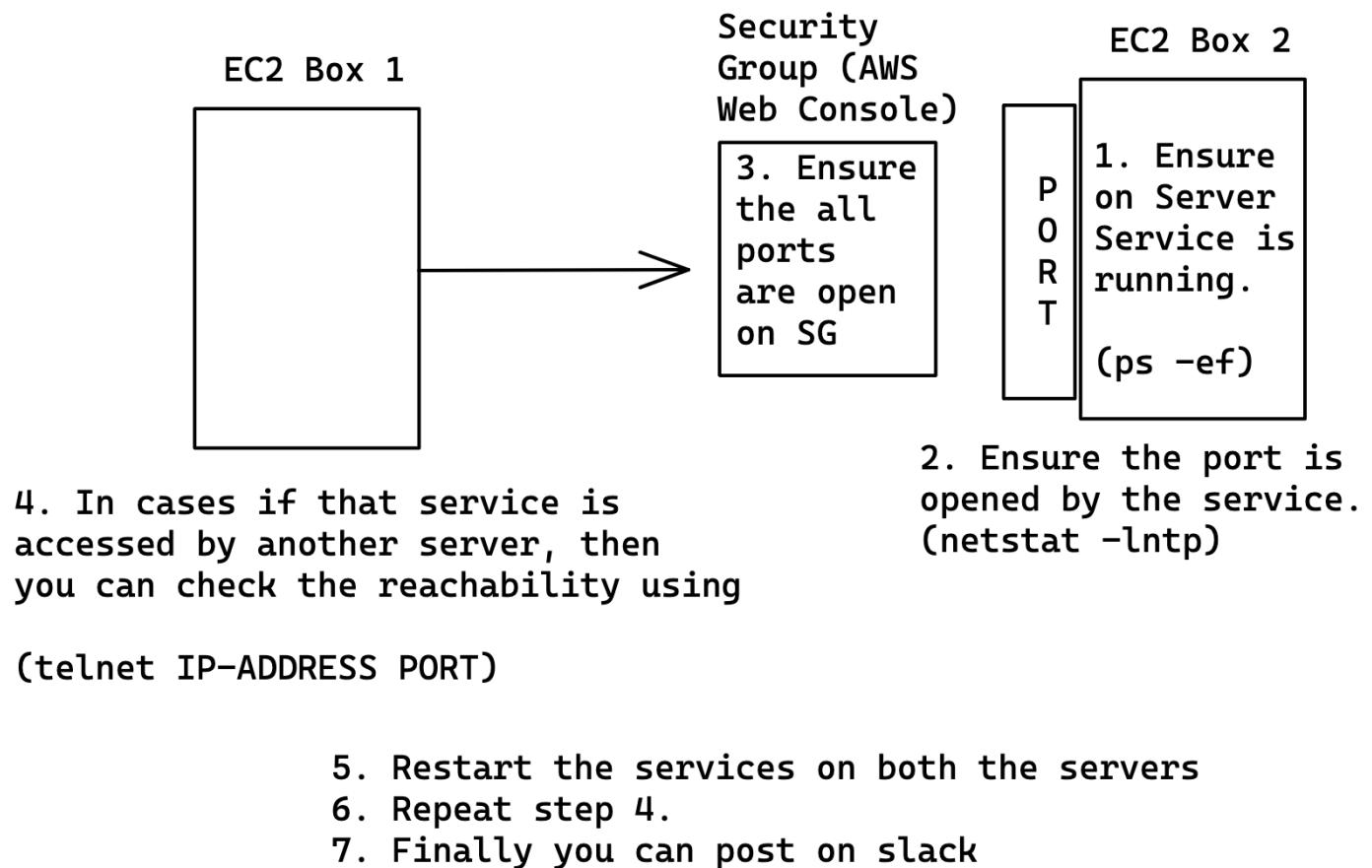
Single - 49.204.179.150/32

Global - 0.0.0.0/0



Generally, we ensure only required ports opened on SG in real time during work. However since we are dealing with LAB we would like to open all the traffic to the server to avoid issues while learning.

Trouble Shooting Guide



Web Based Applications

Web Site
vs
Web Application

Sample Static WebSites

google landing pages

wikipedia pages

news websites

blogging websites

Sample Web Applications

Facebook

Gmail

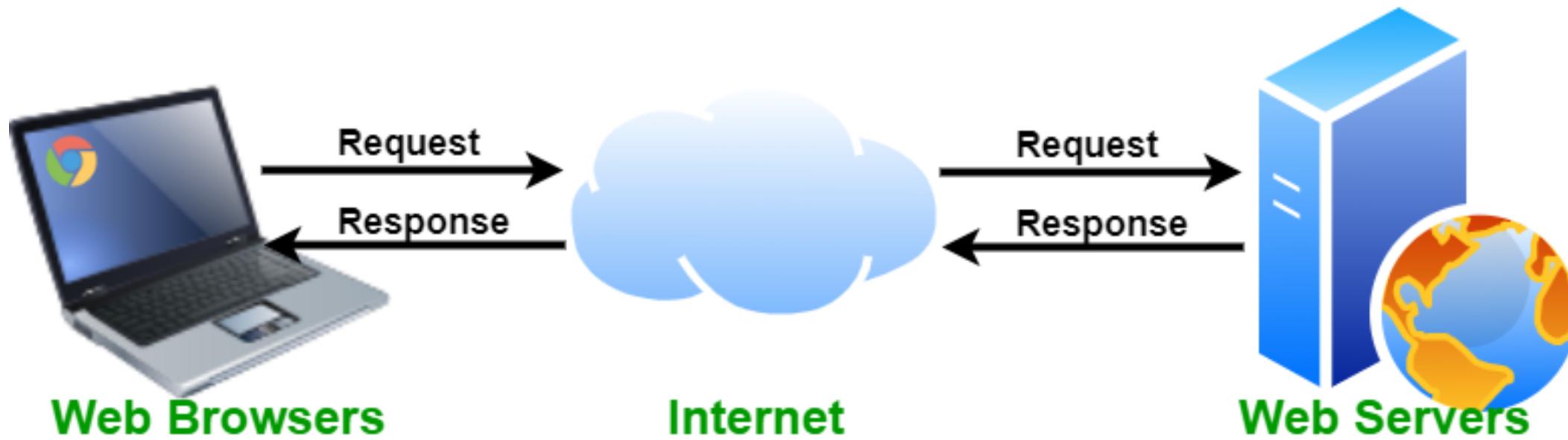
Amazon

Twitter

LinkedIN

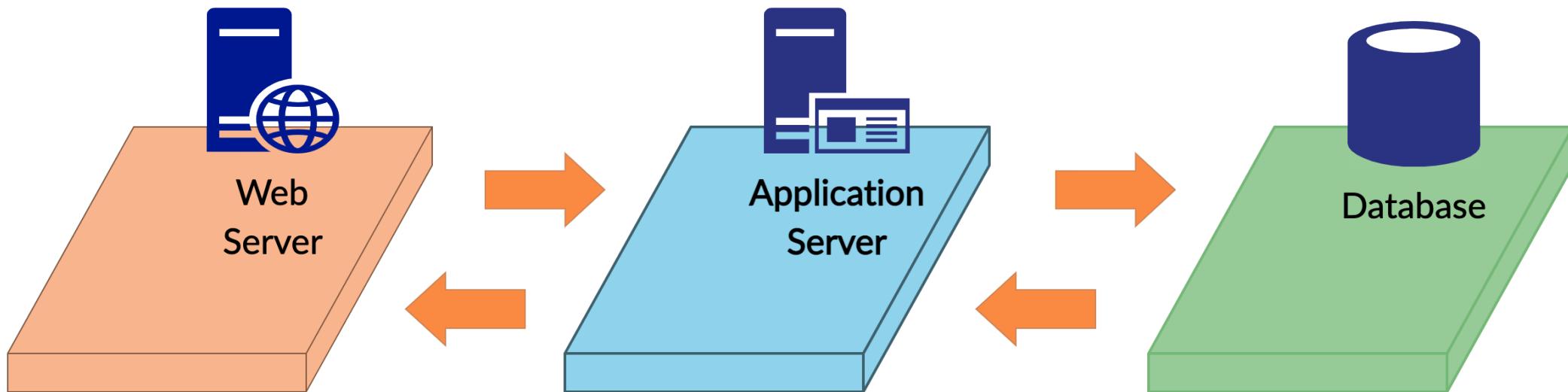
**How you get web content in
Modern Applications?**

WebContent

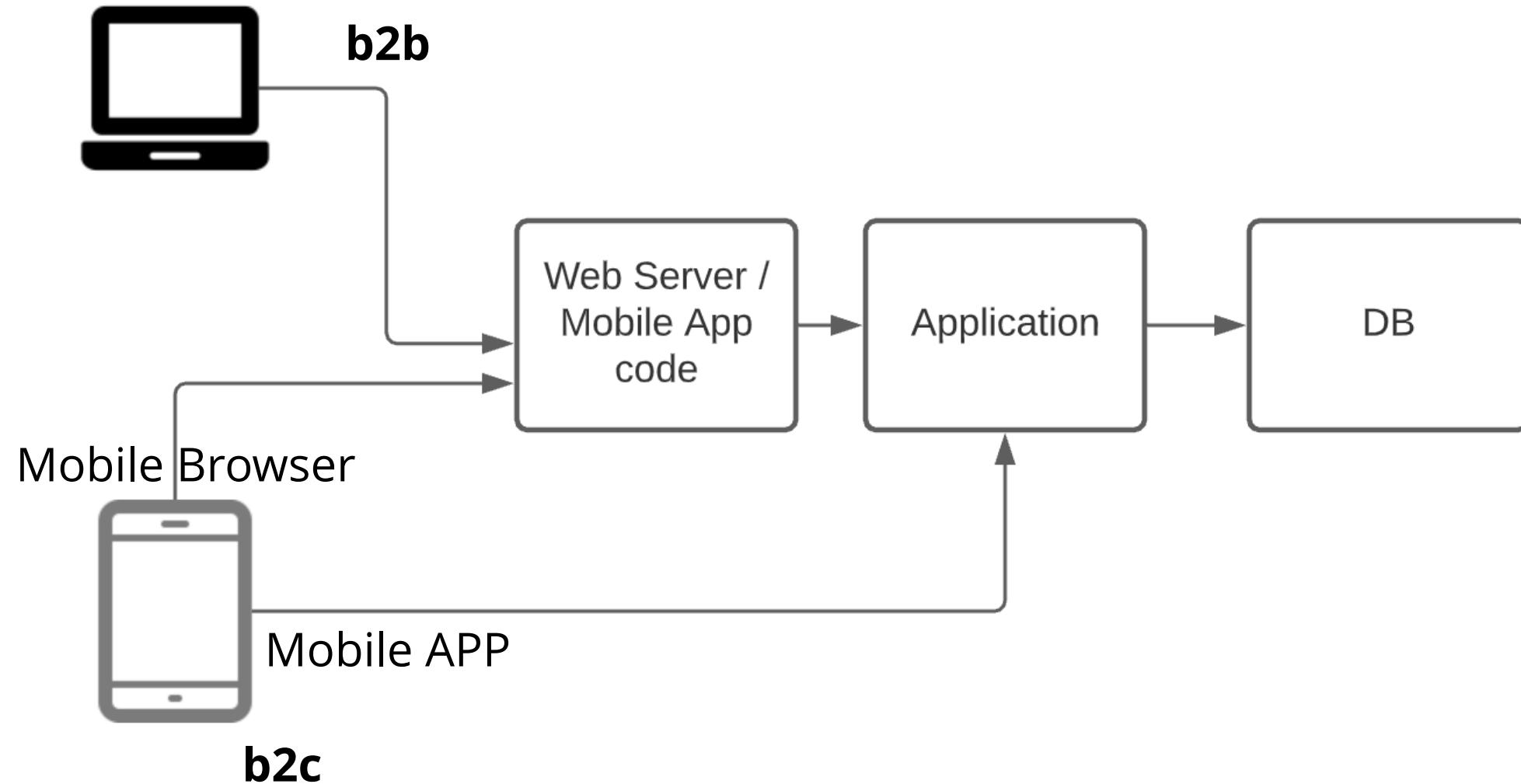


DG

Web Application

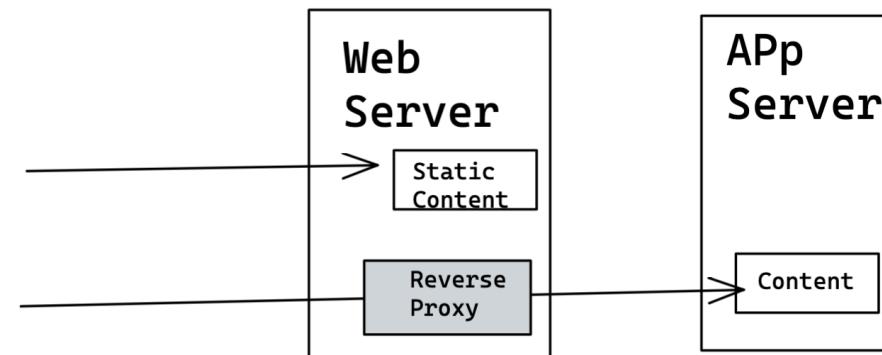


3-Tier Architecture



Web Server

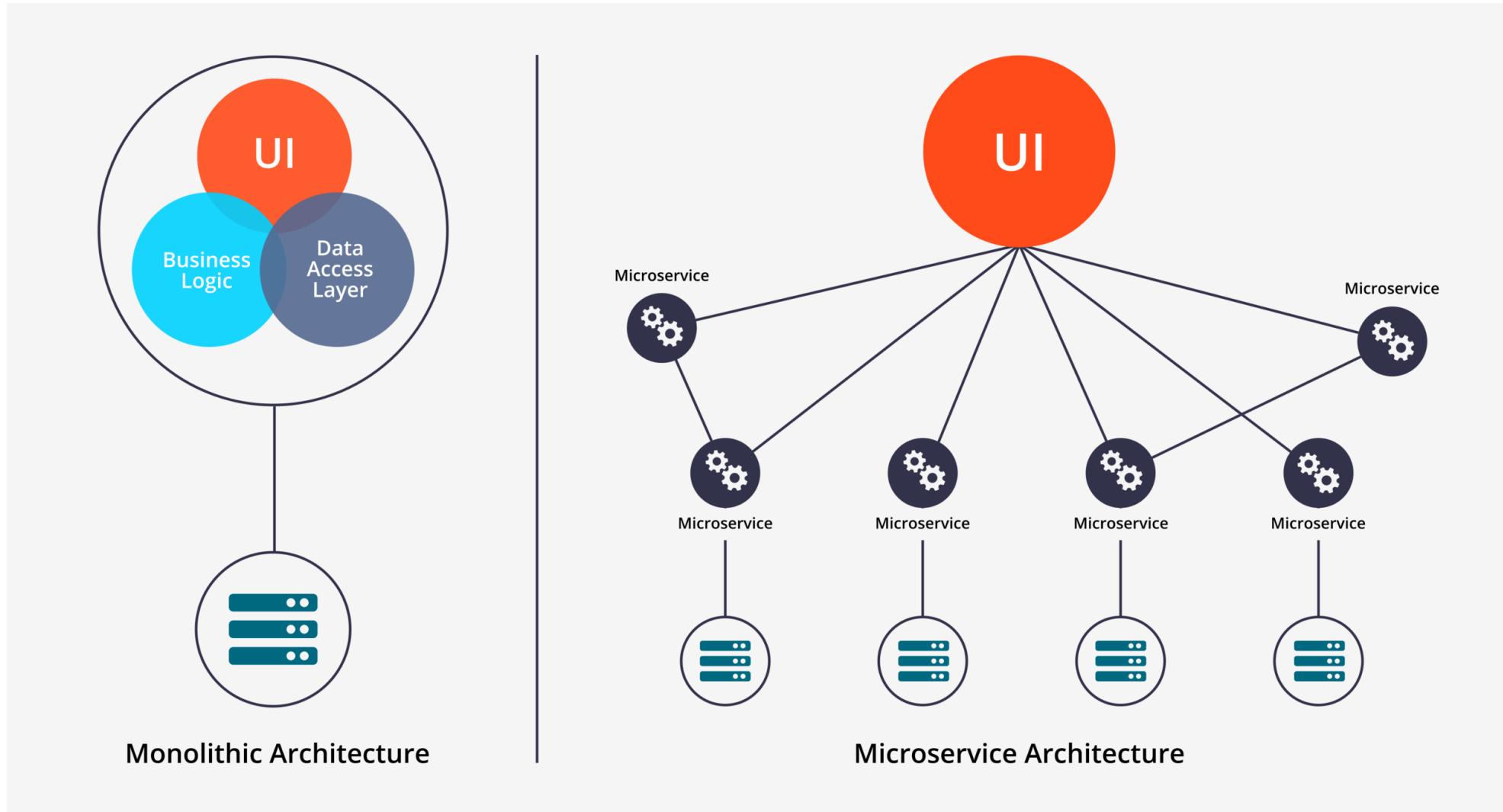
- It can serve the static content
- Reverse Proxy



HTTP Status Codes



Monolith vs Microservices

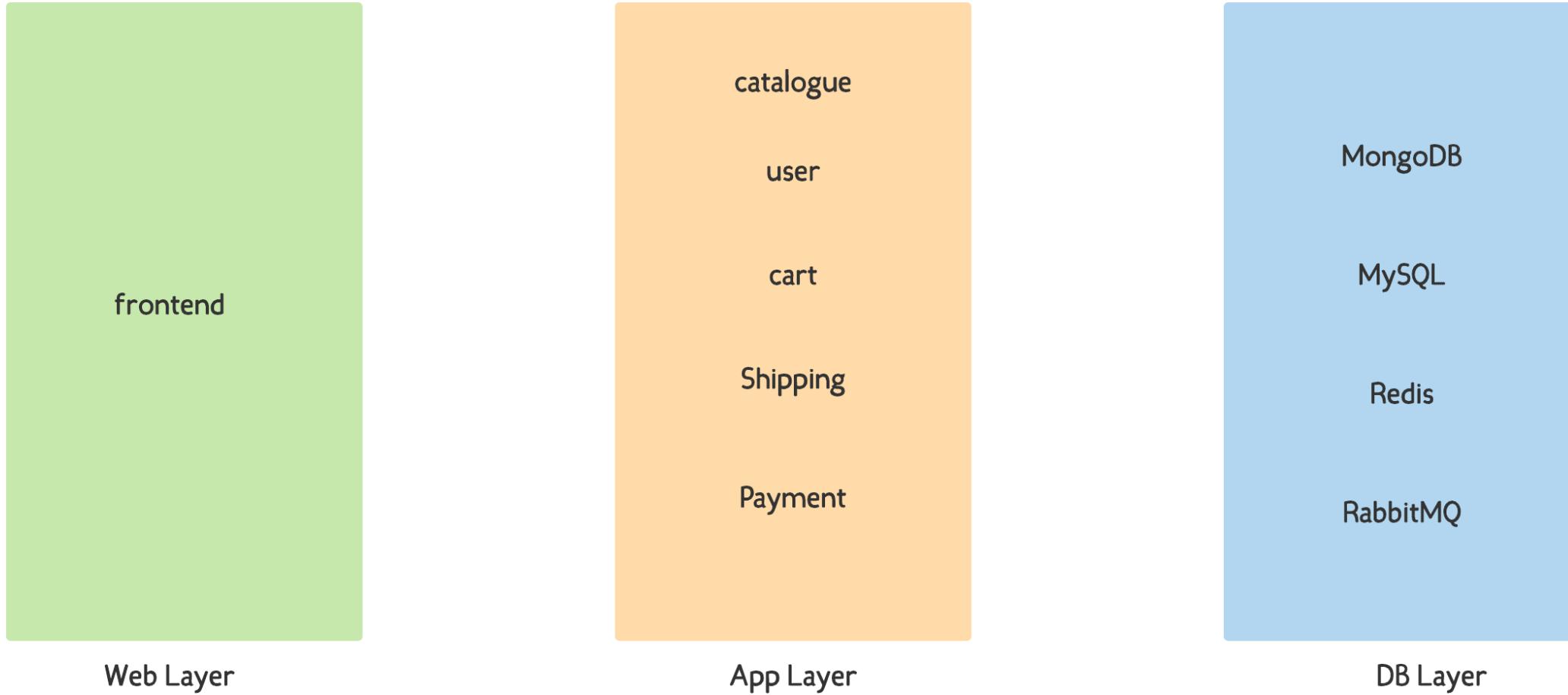


Advantages of MicroServices

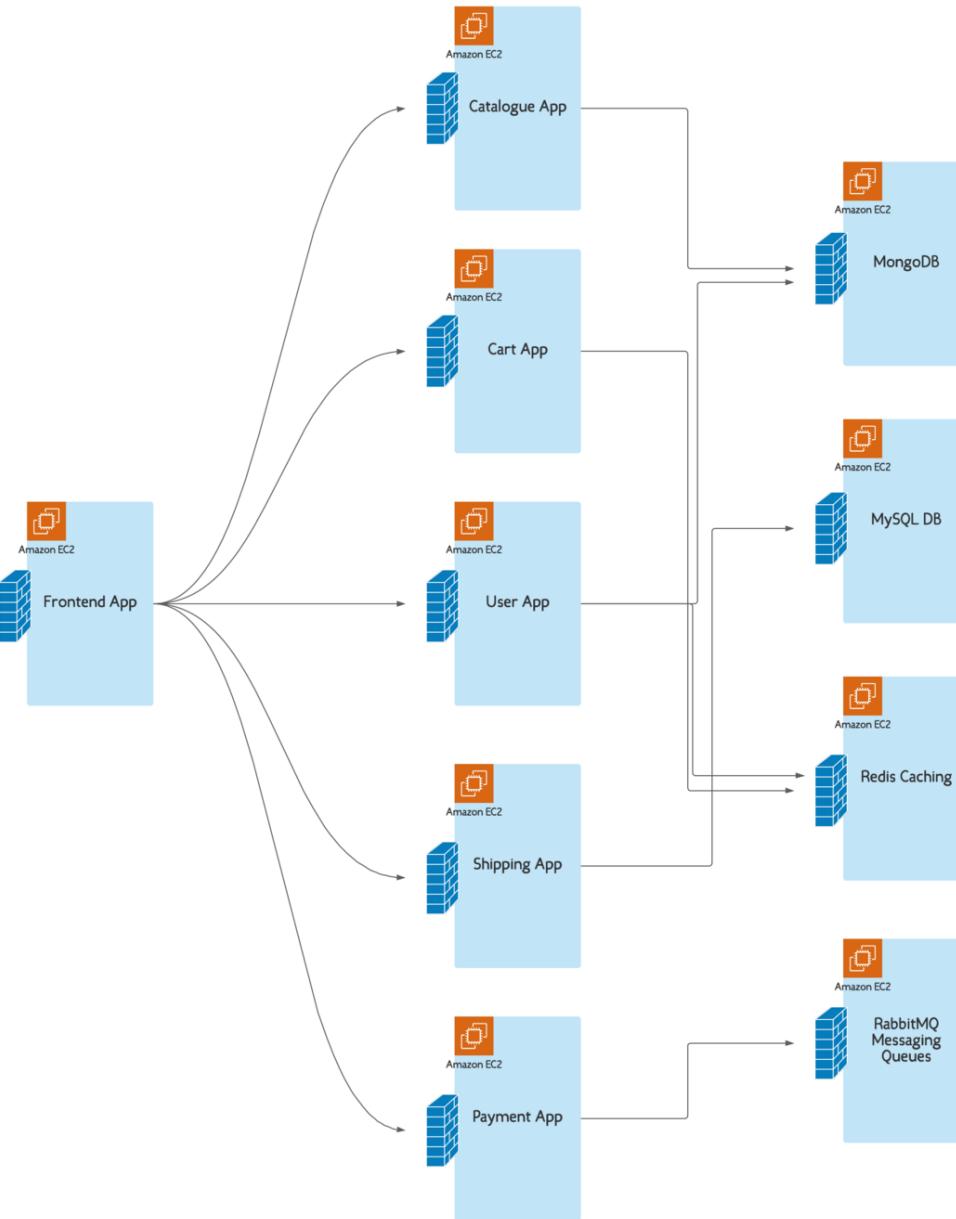
- Impact of the application downtime is very less.
- Horizontal scaling is possible. Helps scaling individual component.
- Helps Agile development as well.
- Helps to build Poly Cloud applications. Each service typically talks over network.
- Microservices also brings flexibility to choose different components (Programming Language & Databases)

**Create One Security Group
& Use that in LAB**

3 Tier Architecture



Roboshop Project Architecture



Public IP vs Private IP

Standards of Project

1. Take individual server for each component. We will not use a single machine for everything, because it is not how the realtime works.
2. We always use private IP address for internal communications.
3. Developers chooses application tech stack (Meaning whether to use java, golang ...)
4. Developers chooses which version of software to be used.

AWS Spot Instances

Stateless Applications

Apps that just power up the business logic and does only compute part but not going to store any data inside them.

Stateful Applications

Apps that are going save the data in HDD, Like Databases.

SPOT we can use for Stateless Apps

AWS Instance Types

t3.micro

t -> AWS Instance Type

3 -> Hardware Refresh number

micro -> Instance size

Instance Types

- On-Demand Instances (\$\$\$)
- Reserved Instances (\$\$)
- Spot Instances (\$)

EC2 Launch Templates

Setup RoboShop Application

Problems we Forecast

DNS Domains

AWS Hosted Zone

Private Hosted Zone

roboshop.internal

This works only in internal/intranet networks.

This is optional

Public Hosted Zone

xyz.com

- This works only in the internet
- This service is costly in AWS, Hence we created a domain in a third party, In our case we used Hostinger.
- Then we gave DNS NS records of AWS in Hostinger.
- This is mandatory

DNS Record Types

A Record (Alias)

A Record is used for denoting an IP address from a name.

CNAME (Canonical Name)

CNAME record is used to denote another name from a name.

Things to consider while creating Private Domain

- Ensure you are using the Private domain while creating it.
- Also add the right region and right VPC so that it will work.

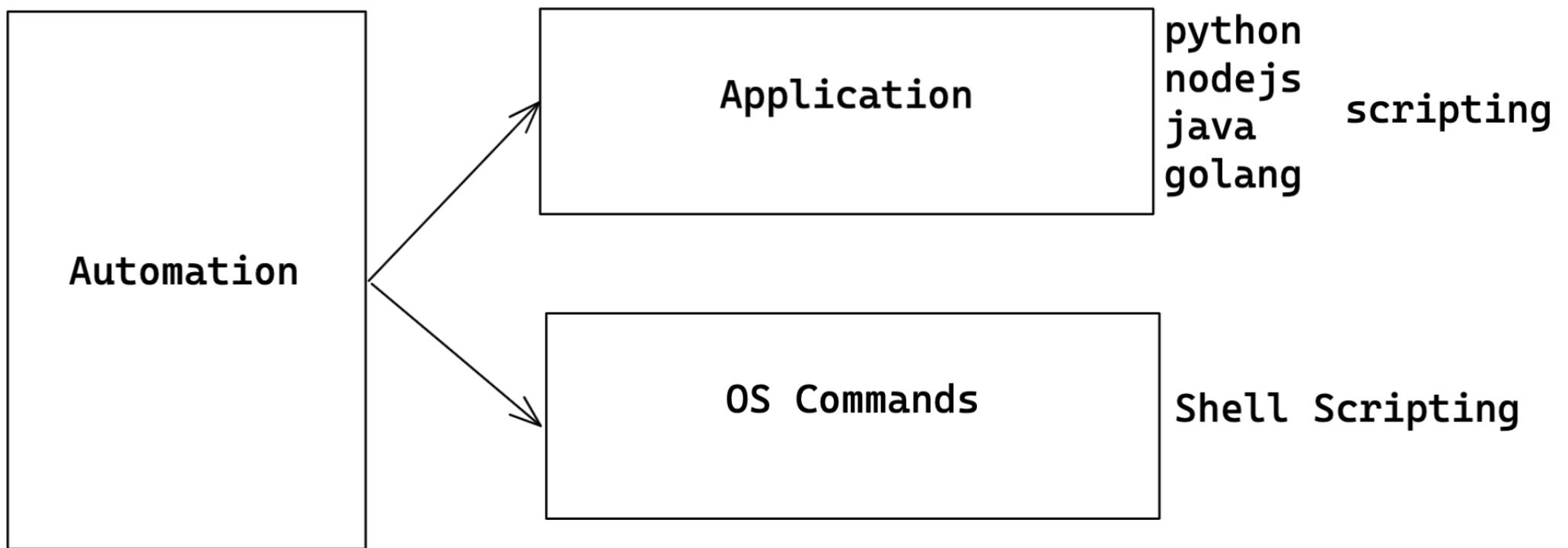
Things to consider while using Private Domain

- Ensure you create a DNS record first and then only use that in configuration of your roboshop project.
- If we use the DNS record without creation and then if we create it will take a longer time to syncup. (May take 24hrs also sometimes)

Manual Approach

What we are automating?

What we want to Automate?



Automation (Scripting)

- **Shell scripting**
- Python scripting
- Ruby Scripting
- Java Scripting
- GoLang
- Type Script

Why Shell?

- Shell is the **default in OS**, No additional installations.
- If we look into the future, Containers are more happening,
Containers are very minimal. You may not get a chance to install
3rd party. Major reason is for security.
- While we go with OS, **Shell is native**, Native always have an
advantage on its performance and behaviour.

When not use Shell?

- While you are not dealing with OS automation then the shell has a limitation.
- While you **deal with applications**, Ex: YouTube, We cannot run with Shell scripting.

Code

Code Standards

- Code should never be kept in local (Laptop/Desktop)
- Always try to push code to central.
- Code should be developed in Local.
- Always choose editors as per your comfort.
- Editors always increase productivity.

Code Best Practices

- Code should never hardcode the **username and passwords**.
- Code **DRY** vs WET, Always try to make code DRY.
- Rerun of automation should never fail.

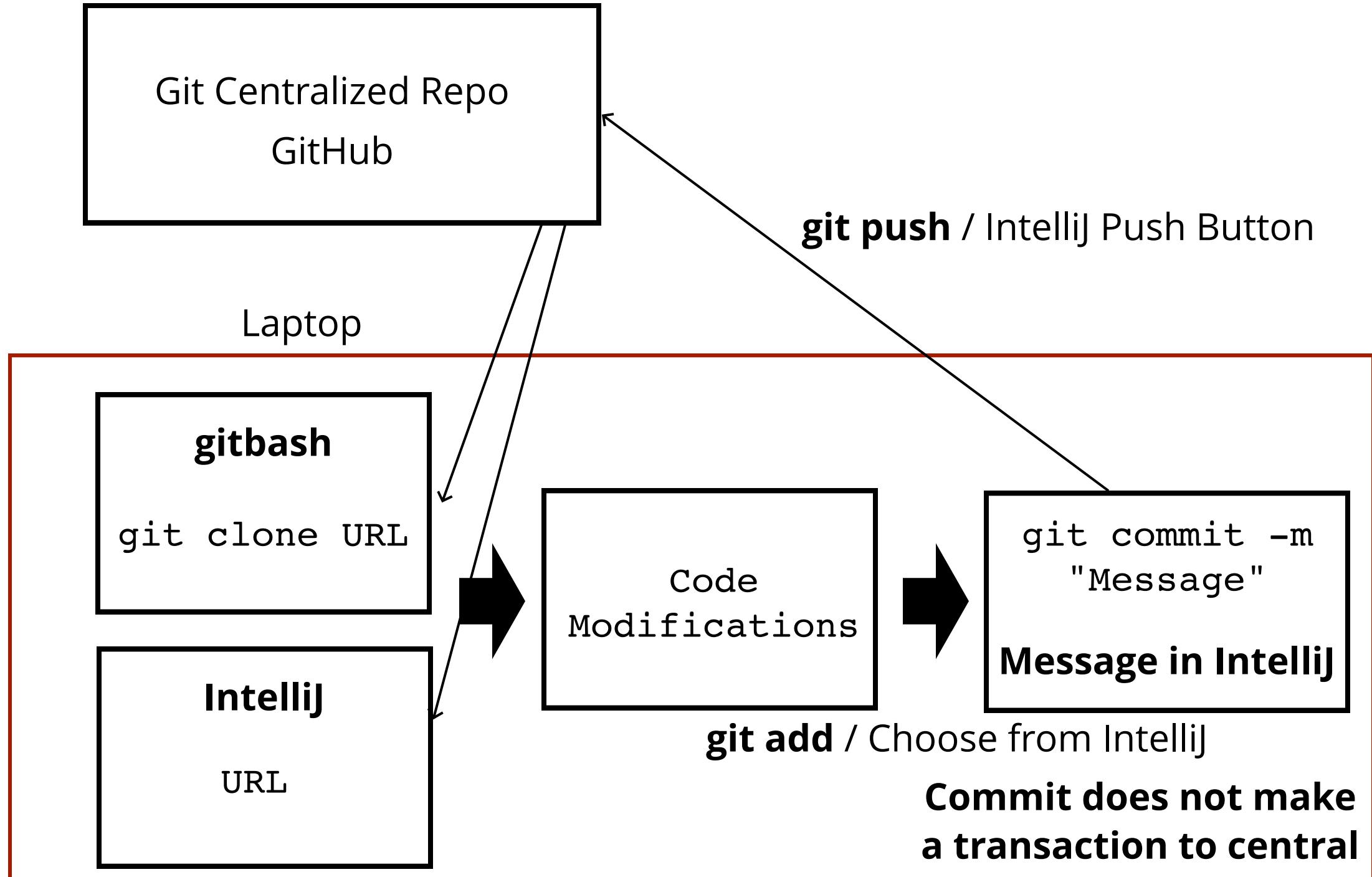
VCS (Version Control System)

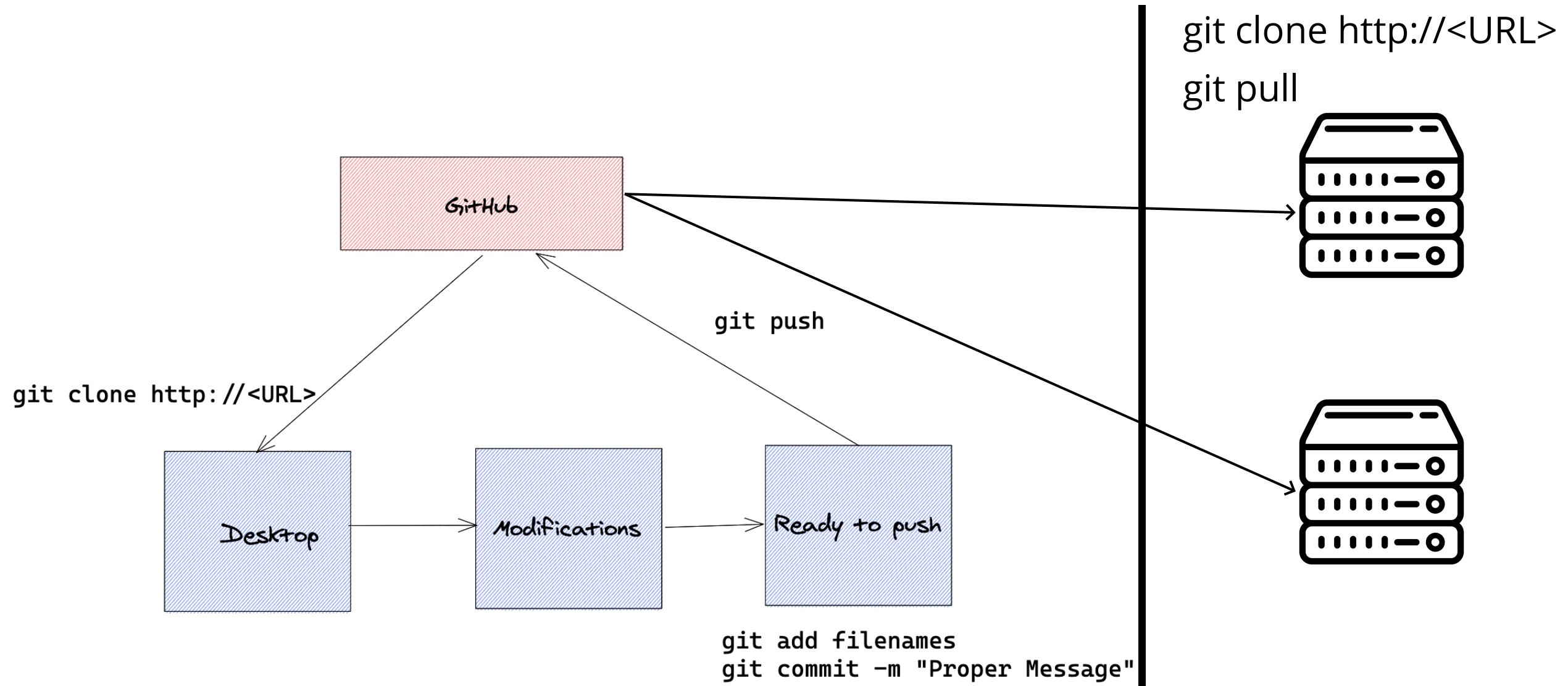
- Maintaining your code in version manner.
- SVN & GIT are famous tools.
- **GIT** is highly used on modern development.

GIT

GitServer in Central?

- Install Git Server in EC2 Like (GitHub / GitLab / Bitbucket)
- AWS Code Commit
- Take public services like (**GitHub** / GitLab/ more)





What happens in Agile?

(GIT Commit Messages)

Shell Scripting

Which Shell

- **BASH**
- SH
- KSH
- CSH
- ZSH
- FISH

Why Bash

- Bash is the default in Linux
- Bash is having all the features of basic shells

She-Bang

- **#!** is called as She-Bang
 - It denotes the path of the interpreter and ensures the remaining lines are executed using that interpreter.
 - For a file only one She-Bang is possible.
 - Also She-Bang has to be in very first line
- ```
#!/bin/bash

This is a sample script
```
- If script executes as <shell> <script> then mentioned shell will be used
  - If the script is executed as ./script then she-bang is used.
  - If the script is executed as ./script but she-bang is not there then default shell bash is used

# Comments

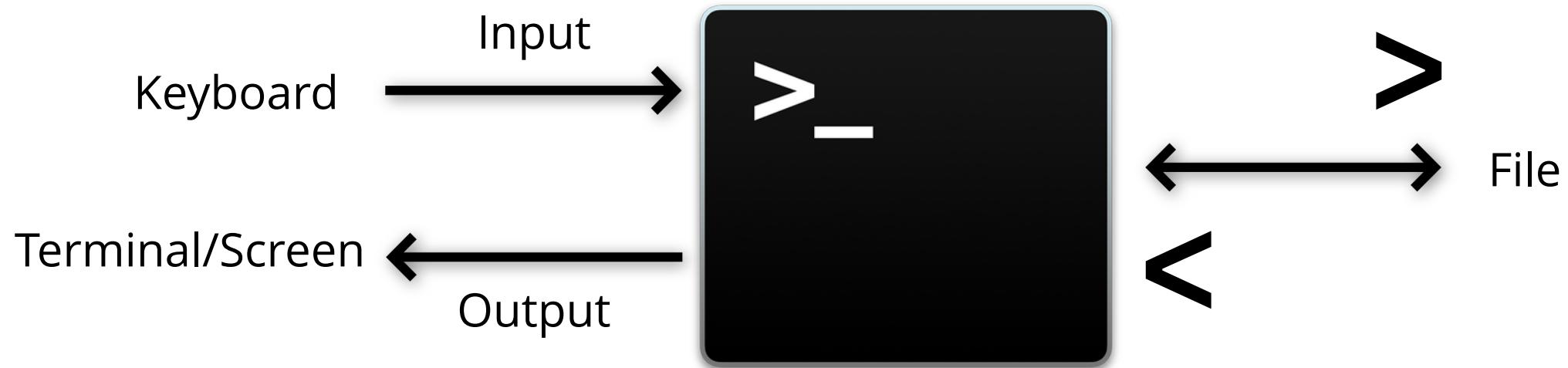
Any line starting with a # character then  
that line will be ignored by the interpreter.

# Quotes

|                      |                                                                                                              |
|----------------------|--------------------------------------------------------------------------------------------------------------|
| <b>Single Quotes</b> | <b>Does not consider any character as a special character</b>                                                |
| <b>Double Quotes</b> | <b>Very few characters like \$ will be considered as special and remaining of them are normal characters</b> |

# Redirectors

(>, <)



# Redirectors

(>, <)

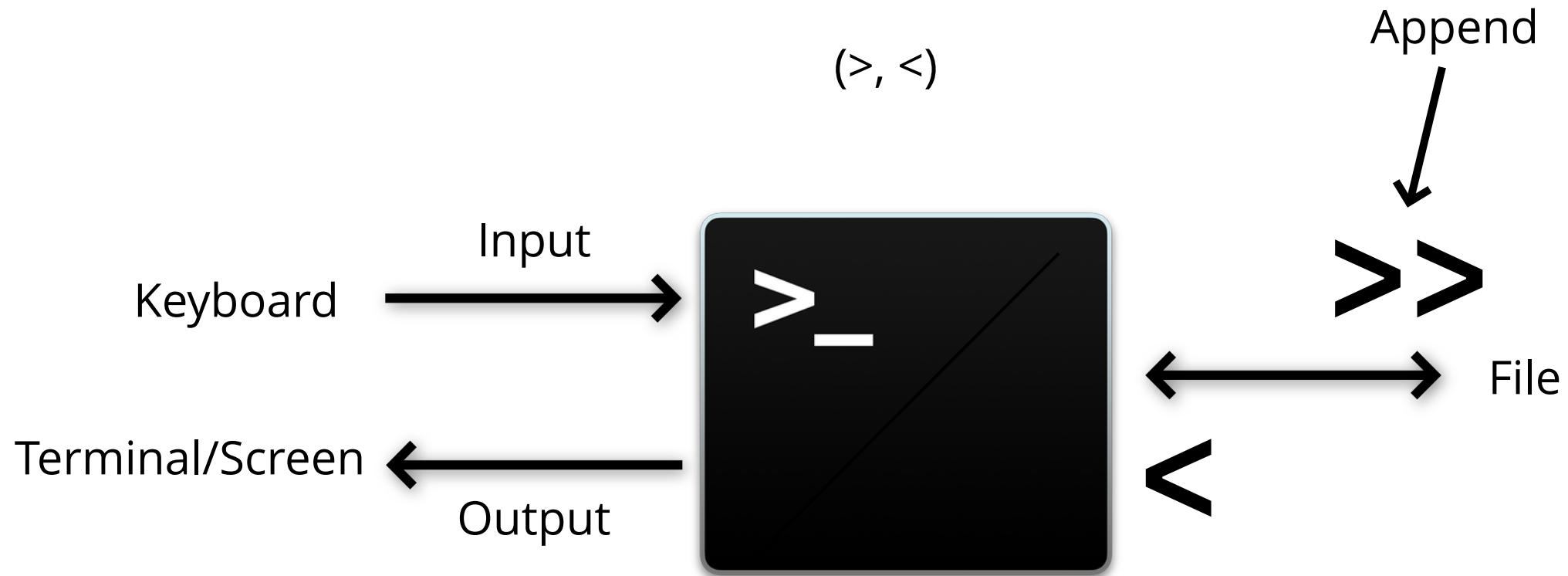
## **STDOUT (>)**

Instead of displaying the output to the screen, if we want to store the output to a file then we use this redirector.

## **STDIN (<)**

Instead of taking the input from keyboard if we want to send through a file then we use this redirector

# Redirectors



# Redirectors

(>, <)

**STDOUT (>)**

Only Output

**STDOUT  
(1>) (>)**

**STDERR  
(2>)**

Only Error

**STDOUT & STDERR (&>)**

Both output and error  
will be redirected to  
the same file

# Redirectors

(>, <)

**&>/dev/null**

In a case if we do not need any kind of output or error to a file for future reference we try to nullify the output with the help of **/dev/null** file

# Print Message

echo command will help us printing message on screen from our scripts.

While printing we can enable some esc sequences for more options

1. \e - To enable Color
2. \n - To print new line
3. \t - To Print New Tab

# Variables

- If you assign a name to set of data that is called as a variable.
- In Bash shell we declare the variable as ***VAR=DATA***
- In Bash Shell we access the variable as ***\$VAR*** or  ***\${VAR}***
- All the time we will not hardcode the value of a variable and we need the data dynamically
- **COMMAND & ARITHMETIC**  
Substitution
- ***VAR=\$(command)*** , this is command subst, Command output will go to VAR variable
- ***VAR=\$((expression))***, this is arithmetic subst, expression output goes to variable. Example is ***\$((2+3))***

# Variables

**VAR=DATA**

- Variable names can have only characters **a-z, A-Z, 0-9, \_(underscore)**
  - Special characters are not allowed
  - A variable should not start with a number and it can start with a Underscore.
  - Variables by default will not have any data types.
  - Everything is a string.
  - As a user you should know that what data would come , since there is no data types.
- 
- In Linux Shell Scripts people from Unix/Linux background considers the variable names will all CAPS. **Ex: COURSENAME**
  - People from Java/DEV background prefer CamelCases Variables **Ex: courseName / CourseName**
  - People from some other backgrounds use snake\_case **Ex: course\_name**

# Variables

- Variables of Bash Shell will hold three properties

## 1. ReadWrite

```
$
$ a=10
$ echo $a
10
$ a=20
$ echo $a
20
$
```

## ReadOnly

```
$ a=100
$ readonly a
$ a=200
-bash: a: readonly variable
$
```

## 2. Scalar

```
$
$ a=10
$ echo $a
10
$ a=20
$ echo $a
20
$
```

## Arrays

```
$ b=(10 20)
$ echo ${b[0]}
10
$ echo ${b[1]}
20
$
```

## 3. Local

```
$ vi /tmp/1.sh
$ cat /tmp/1.sh
#!/bin/bash

echo $a
$ a=100
$ sh /tmp/1.sh

$
```

## Environment

```
$ a=100
$ sh /tmp/1.sh

$ export a
$ sh /tmp/1.sh
100
$
```

# Inputs

- Most of the times we cannot hardcode the values and also at the same time we cannot dynamically determine the values using Command Subst.
- So we need input from the user and based on input, we proceed with the script.
- Input in shell scripting can be taken in two ways.
  1. **During Execution**
  2. **Before execution.**

# Inputs - During Execution

- While executing the script we ask the input from the user using **read** command.
- This approach will be taken if for sure we know that script will be executed manually.
- Otherwise, this approach will not work for automation. It breaks the automation and ends with failures.

# Inputs - Before Execution

- Before executing the script we can provide the inputs and those inputs can be loaded by script.
- Those values will be loaded by **Special Variables** inside the shell.
- This is the approach will be taken by most of the commands in the shell.
- Special variables are **\$0-\$n, \$\*, \$@, \$#**.
- \$0 - Script Name
- \$1 - \$n - Arguments parsed in the order.
- \$\*, \$@ - All arguments
- \$# - Number of arguments

# Inputs - Before Execution

**Example: script1.sh 10 abc 124**

| Special Variable | Purpose (To get what) | Values from Example |
|------------------|-----------------------|---------------------|
| \$0              | Script Name           | script1.sh          |
| \$1              | First Argument        | 10                  |
| \$2              | Second Argument       | abc                 |
| \$*              | All Arguments         | 10 abc 124          |
| \$@              | All Arguments         | 10 abc 124          |
| \$#              | Number of Arguments   | 3                   |

# Exit Status

- When we don't have any kind of output from the executed command and if we want to determine whether that command is executed successfully or not then we can refer the Exit States.
- Not only the above scenario most of the automation around scripting deals with exit status only.
- Exit status ranges from 0-255
- 0 - Universal Success
- 1-255 is Not Successful / Partial Successful

# exit Command

- exit command by default return exit status 0 (zero)
- Also exit status stops the script and will not execute any other commands.
- exit command is free to use any number from 0-255.
- But the values from 125+(126-255) are usually used by the system, Hence we do not use those values in the scripts.
- Hence Script Author is always recommended to use the values from 0-125

# Functions

- If you assign a name to set of commands then that is a function.
- Functions are used to keep the code **DRY** (reusability).
- Functions are also used to group the commands logically.

# Variables with Functions

- If we declare a variable in the main program then you can access them in function and vice-versa.
- Variables of the main program can be overwritten by function and vice-versa.
- Special variables which access the arguments from CLI cannot be accessed by function. Because functions will have their own arguments.

# exit(return) Status Functions

- In times we need to come out of function to the main program, but we cannot use **exit** command because exit will completely exit the script.
- **return** command will be used to come out of function.
- Function is also a type of command which necessarily needed the exit status. So **return** command is capable of returning status like **exit** command and number ranges from 0-255.

# Project Code Structure

1. Individual scripts for all components of the project.
2. All components are written in one script.

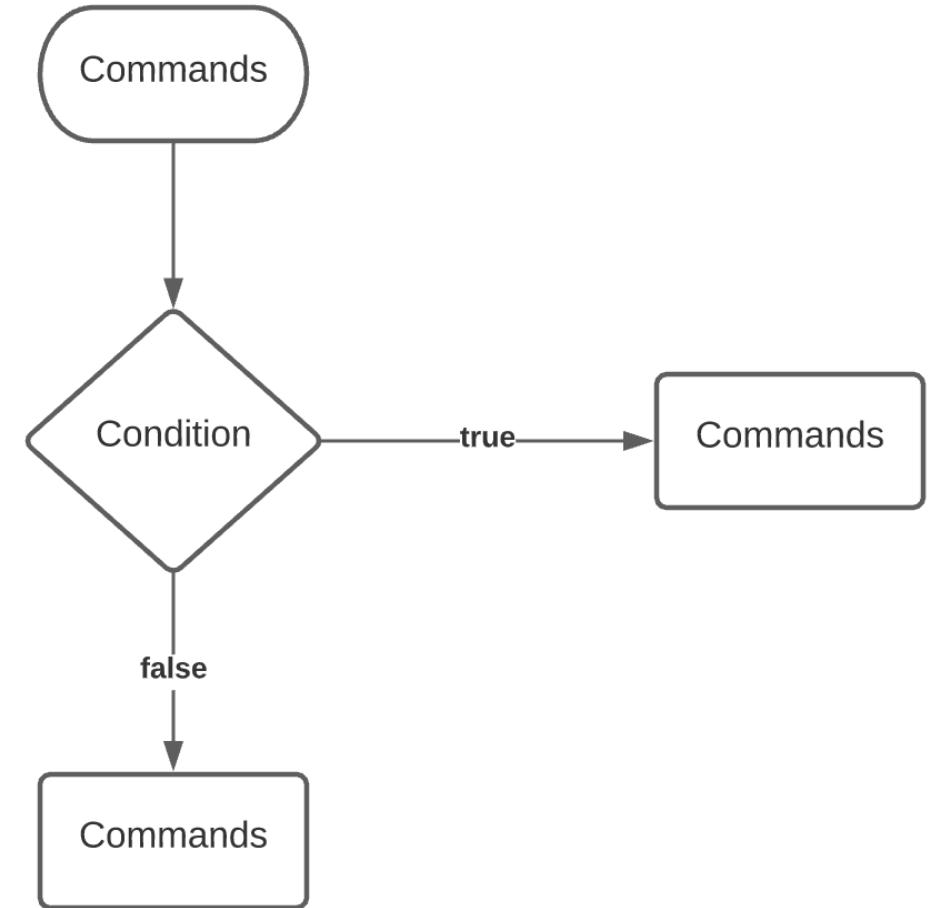
# Best Structure for Project

1. Individual scripts for all components wrapped with own shell script.
2. Individual scripts for all components wrapped with Makefile

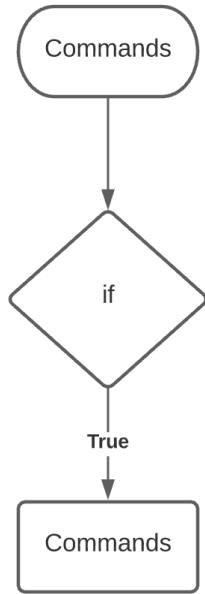
**Title Text**  
**Subtitle**

# Conditions

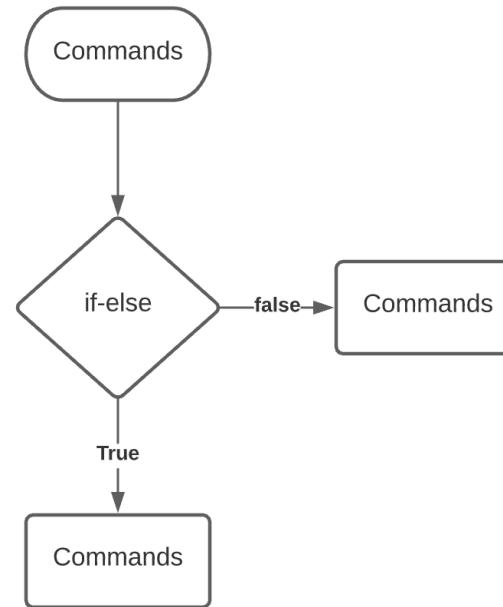
- case
- if



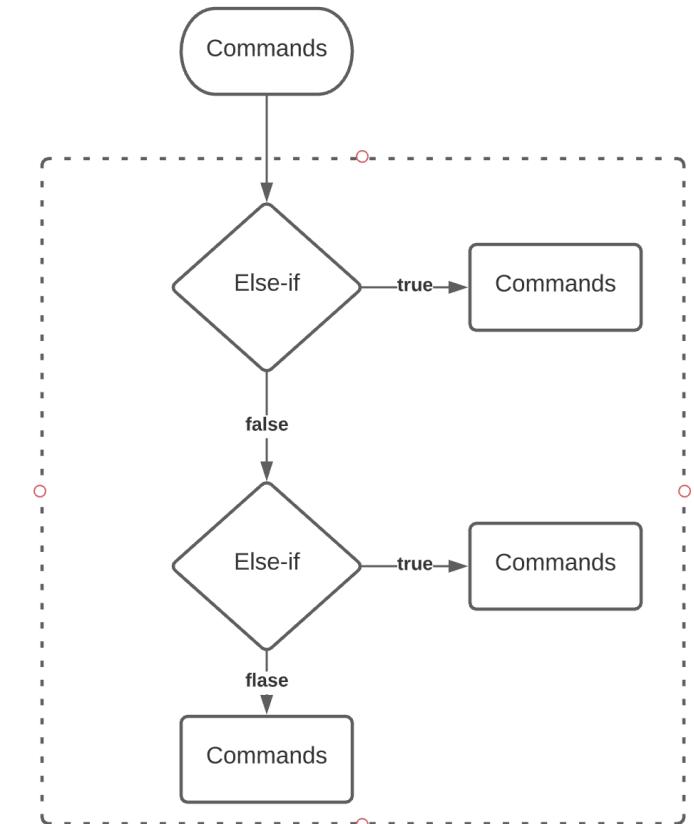
# IF condition



Simple IF



If Else



Else If

# IF condition

## Simple IF

```
if [expression]
then
 commands
fi
```

## If Else

```
if [expression]
then
 commands
else
 commands
fi
```

## Else If

```
if [expression1]
then
 commands1
elif [expression2]
then
 commands2
elif [expression3]
then
 commands3
else
 commands4
fi
```

# Expressions

From the previous, if type syntaxes, You can observe that all the conditions are dependent on expressions. Lets categorize them in three.

## String Comparision

**Operators:** = , ==, !=, -z

### Examples

```
["abc" == "ABC"]
```

```
["abc" != "ABC"]
```

```
[-z "$USER"]
```

## Number Comparision

**Operators:** -eq, -ne, -gt,  
-ge, -lt, -le

### Examples

```
[1 -eq 2]
```

```
[2 -ne 3]
```

```
[2 -gt 3]
```

```
[2 -ge 3]
```

```
[2 -lt 3]
```

```
[2 -le 3]
```

## File Comparision

**Operators:** -f, -e

### Examples

```
[-f file]
```

```
[! -f file]
```

```
[-e file]
```

# Logical AND & Logical OR

`command1 && command2`

`&&` symbol is referred as Logical AND, command2 will be executed only if command1 is successful.

`command1 || command2`

`||` symbol is referred as Logical OR, command2 will be executed only if command1 is failure.

# **SED (Stream Line Editor)**

- Delete the lines.
- Substitute a word
- Add the lines.

# SED (Stream Line Editor)

SED command works in two modes depends up on the option you choose.

1. sed will not change file and print the changes on the terminal, this is the default behaviour of SED.
2. If we need to edit the file rather than just printing on the screen then we have to use **-i** option

# SED (Stream Line Editor)

Delete the Lines

```
sed -e '/root/ d' /tmp/passwd
```

```
sed -i -e '/root/ d' /tmp/passwd
```

```
sed -i -e '/root/ d' -e '/nologin/ d' /tmp/passwd
```

```
sed -i -e '1 d' /tmp/passwd
```

# SED (Stream Line Editor)

Substitute the Words

```
sed -e 's/root/ROOT/' /tmp/passwd
```

```
sed -i -e 's/root/ROOT/' /tmp/passwd
```

```
sed -i -e 's/root/ROOT/gi' /tmp/passwd
```

# SED (Stream Line Editor)

Add the new lines

```
sed -e '1 i Hello' /tmp/passwd
```

```
sed -i -e '1 i Hello' /tmp/passwd
```

```
sed -i -e '1 a Hello' /tmp/passwd
```

```
sed -i -e '1 c Hello' /tmp/passwd
```

```
sed -i -e '/shutdown/ c Hello' /tmp/passwd
```

# Case Condition

- case is almost like else-if
- case can do only string comparison but will not be able to compare numbers or files.

```
case $var in
 pattern1) commands1 ;;
 pattern2) commands2 ;;
 *) commands ;;
esac
```

# Loops

- Fundamentally we have two loop commands, **for & while**.
- If inputs are known then go with for loop.
- If input can be controlled then go with while loop

# Completed Project with Shell Scripting

# In Gen

**dev**

**qa**

**stage**

**uat**

**cit**

**pre-prod**

**nonprod**

**prod / live**

**perf**

**dr**

**sandbox**

# Multi Env

**dev**

**qa**

**prod**

# LAB

# Problems we can Forecast

- **Automated using scripting**
- Standalone Servers
- **Dependency Configuration solved DNS**
- Scalability
- Security

# Problems with Shell Scripting

- **Imperative** vs Declarative
- **Homogeneous** vs Heterogeneous
- **Sequential Operations** Vs Parallel Operations
- **Script has to be local on Server**

```
Declarative Style

user sample
```

```
Identify OS
UNAME=$(uname)
case $UNAME in
 Linux)
 grep REDHAT /etc/os-release &>/dev/null
 if [$? -eq 0]; then
 useradd sample
 fi
 grep DEBAIN /etc/os-release &>/dev/null
 if [$? -eq 0]; then
 adduser sample
 fi
 ;;
 SunOS)
 ;;
 HPUX)
 ;;
 AIX)
 ;;
esac
```

# Configuration Management Tools

# CM Tools

- Puppet
- Chef
- Salt
- **Ansible**

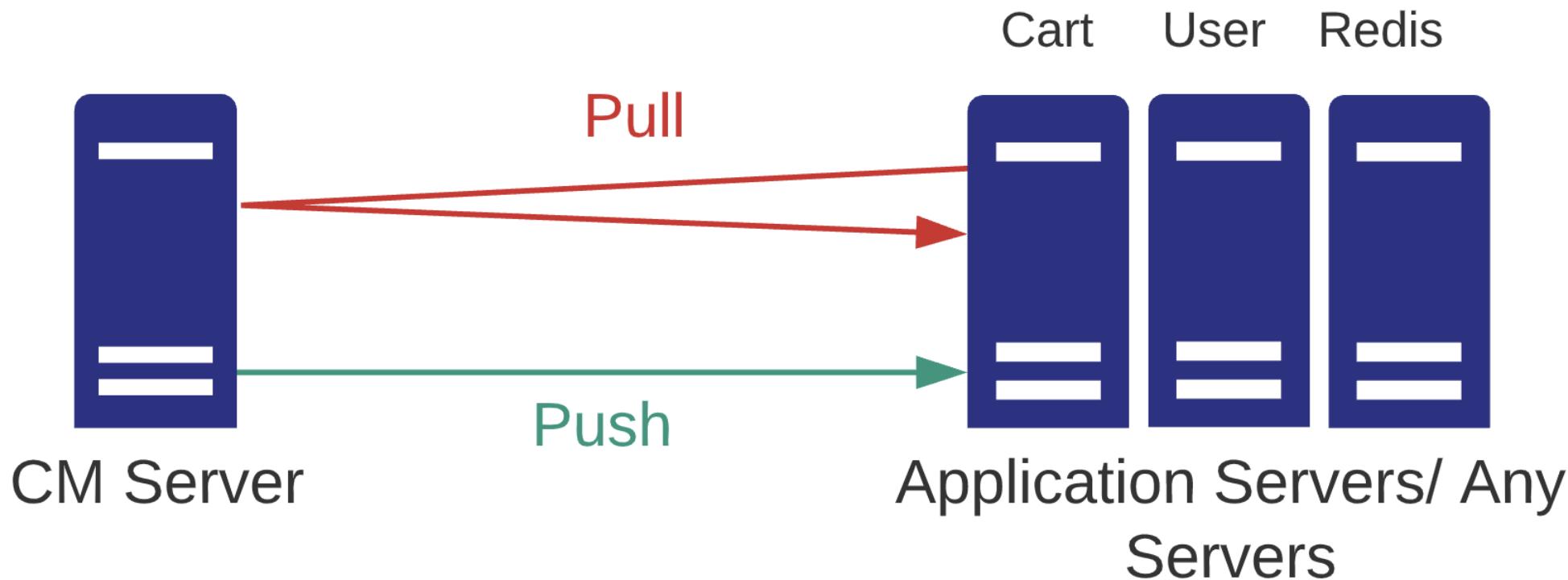
# Ansible

# Ansible Solves

- Imperative replacing with **Declarative**
- Simple declarative supports **heterogeneous OS**
- Parallel operations are **possible**
- Code need **not to be** on the server

# Ansible

# Differences b/w Pull & Push



# Push

- Once you push and changes made later manually on server are not been taken care

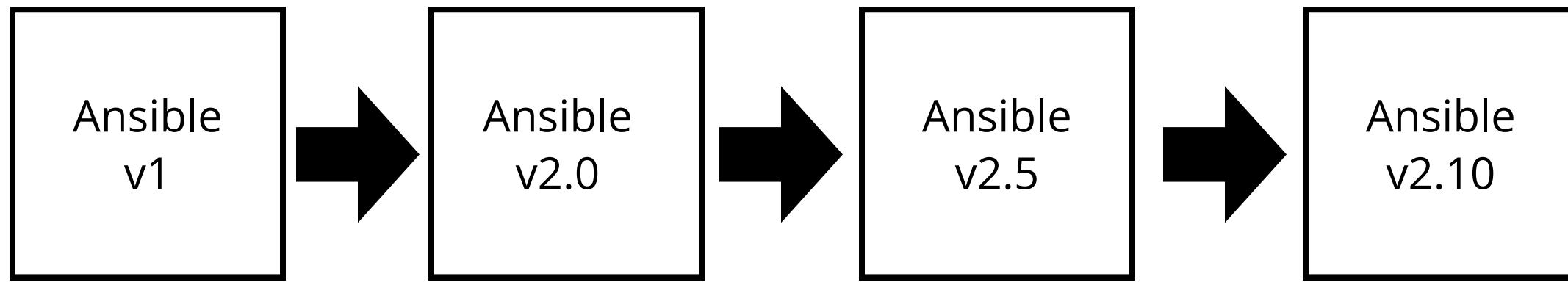
# Pull

- Pull will happen by an additional agent runs on the client machines, Meaning one more software run and involves operations.
- What if that agent is crashing ? Pull never gives centralized report.

**Ansible offers both push and pull mechanism.**

# Ansible Versions

## Ansible Core



---

Until 2.9 it was called as module

Collections

- Ansible (RedHat) 3 -> Ansible Core 2.10
- Ansible (RedHat) 4 -> Ansible Core 2.11
- Ansible (RedHat) 5 -> Ansible Core 2.12
- Ansible (RedHat) 6 -> Ansible Core 2.13

# Ansible 2.9

## Module Index

- All modules
- Cloud modules
- Clustering modules
- Commands modules
- Crypto modules
- Database modules
- Files modules
- Identity modules
- Inventory modules
- Messaging modules
- Monitoring modules
- Net Tools modules
- Network modules
- Notification modules
- Packaging modules
- Remote Management modules
- Source Control modules
- Storage modules
- System modules
- Utilities modules

```
- name: Copy file with owner and permissions
copy:
 src: /srv/myfiles/foo.conf
 dest: /etc/foo.conf
 owner: foo
 group: foo
 mode: '0644'
```

module:

**yum install ansible -y**

# >= Ansible 2.10

## Collection Index

These are the collections with docs hos

- amazon.aws
- ansible.builtin
- ansible.netcommon
- ansible posix
- ansible.utils
- ansible.windows
- arista.eos
- awx.awx
- azure.azcollection
- check\_point.mgmt
- chocolatey.chocolatey
- cisco.aci
- cisco.asa
- cisco.dnac
- cisco.intersight
- cisco.ios
- cisco.iosxr
- cisco.ise
- cisco.meraki
- cisco.mso
- cisco.nso
- ...

```
- name: Copy file with owner and permissions
ansible.builtin.copy:
 src: /srv/myfiles/foo.conf
 dest: /etc/foo.conf
 owner: foo
 group: foo
 mode: '0644'
```

collection.module:

**pip3 install ansible**

# Inventory

- We can keep either IP address or Hostname.
- Grouping can be done either individual files based on environment or based on component and even together and that always depends upon the architecture design that project had.
- In environments like a cloud where the nodes are too dynamic and where your IP addresses always change frequently, we need to work on some dynamic inventory management.

# Ansible Connections

- Ansible uses ssh credentials in the background.

# Ansible Push

# Ansible Modules (Collections)

# Problems of Ansible Ad-Hoc

1. Multiple tasks
2. Logical Limitations

# Introduction to Ansible Playbook

# Playbook

Markup Language will help in sharing the information between systems. **That been extended the sharing of info from user to system or system to user.**



XML

JSON  
YAML

# Any Markup Language

Key -Value - Plain

Key -Multiple Values. - List

Key - Key-Value - Map

# XML eXtensible Markup Lang

```
1 <courseName>DevOps</courseName>
2 <trainerName>Raghu K</trainerName>
3 <timings>
4 "0600IST",
5 "0730IST"
6 </timings>
7 <topics>
8 <aws>
9 "EC2",
10 "S3"
11 </aws>
12 <devops>
13 "Ansible"
14 </devops>
15 </topics>
16 <phoneNumbers>
17 <personal>999</personal>
18 <mobile>888</mobile>
19 </phoneNumbers>
```

# JSON- Java Script Object Notation

```
1 {
2 "courseName": "DevOps",
3 "trainerName": "Raghu K",
4 "timings": [
5 "0600IST",
6 "0730IST"
7],
8 "topics": {
9 "aws": [
10 "EC2",
11 "S3"
12],
13 "devops": ["Ansible", "Shell Scripting"]
14 },
15 "phoneNumbers": { "personal": 999, "mobile": 888 }
16 }
```

# YAML - Yet Another Markup Lang

```
1 courseName: "DevOps"
2 trainerName: "Raghu K"
3 timings:
4 - 0600IST
5 - 0730IST
6 topics:
7 aws:
8 - EC2
9 - S3
10 devops: ["Ansible", "Shell Scripting"]
11 phoneNumbers: { personal: 999, mobile: 888 }
```

1. Indentation is the way of YAML inputs provided.
2. Always use uniform spacing.
3. Tab space are not allowed
4. Based on the program you are dealing, Keys are provided by program and we have to fill those values as per our requirement.
5. Some cases we create our own keys, Mainly like Variables
6. Some cases the values also will be predefined and we have to choose only one of them

# **YAML File Extension**

**.yaml**

**.yml**

# Ansible Playbook

- Playbook has multiple plays.
- Playbook file itself is a list.
- Playbook book should have at least one play.
  - Play must have information about the inventory group. (**hosts**)
  - It should have the information that it should load **tasks** or **roles**.
  - In general, we provide an optional key **name** to denote the purpose of the play & task.

```
1 - hosts: DATABASES
2 tasks:
3 - ansible.builtin.debug:
4 msg: "Hello World"
5
6 - name: Play 2
7 hosts: APPLICATION
8 roles:
9 - roleA
10 - roleB
```

- In the example above **debug** is a module. **msg** is a parameter from debug module.

# When to Quote for Values

1. Ansible Values supports Double Quotes & Single Quotes.
2. In Double Quotes & Single Quotes we can access variables.
3. Quotes are necessary, if the value starts with variable then we have to quote it, Otherwise quotes are not necessary.

# Ansible Variables

## Pre-Defined from User

### (Hardcoded)

- Play Level Variables
- Variables from a file
- Task Level Variables
- Inventory file Variables
- Command Line Variables

# Variable Precedence

In the following order, variables are prioritized, Order is high to low.

- **Command line variables**
- Task Level Variable
- Variable from files
- Play level variable
- Inventory variables

```
- name: Variable
hosts: all
vars:
 URL: vars.example.com 4
tasks:
 - name: Print URL
 ansible.builtin.debug:
 msg: URL = {{ URL }}

 - name: Print URL from tasks
 ansible.builtin.debug:
 msg: URL = {{ URL }}
 vars:
 URL: tasks.example.com 2

- name: Variable from files
hosts: all
vars_files:
 - sample-vars.yml 3
tasks:
 - name: Print URL
 ansible.builtin.debug:
 msg: URL = {{URL}}

- name: Variable from Inv
hosts: all
tasks:
 - name: Print URL
 ansible.builtin.debug:
 msg: URL = {{URL}}
```

5 \$ ansible-playbook -i inventory -u centos -k  
02-vars.yml -e URL=cli.example.com 1

---

# **Ansible Pre-Defined Variables (Facts)**

# Ansible Run Time Variables

- From a task output (**register**)
- Set a variable using task (**set\_fact**)

# Roles

```
playbooks
site.yml
webservers.yml
fooservers.yml
roles/
 common/
 tasks/
 handlers/
 library/
 files/
 templates/
 vars/
 defaults/
 meta/
 webservers/
 tasks/
 defaults/
 meta/
```

# Variable Precedence while roles are used

In the following order, variables are prioritized, Order is high to low.

- **Command line variables**
- Task Level Variable
- ***vars dir from roles***
- Variable from files
- Play level variable
- Inventory variables
- ***defaults dir from roles***

# Variable Data Types

URL7: vars.google.com

NUMBER: 7

NUM1: 7.7

BOOL1: true

BOOL2: false

BOOL3: yes

BOOL4: no

STRING: "true"

NUMBER2: "7"

# Conditions (*when*)

# Loops (*loop*)

# RoboShop Project Structure

- Single Play with all the tasks/roles loaded.
- Multiple Plays each doing the job of individual component

Tags  
Handlers  
Vault

# Parameter & Secret Management

# General Tools

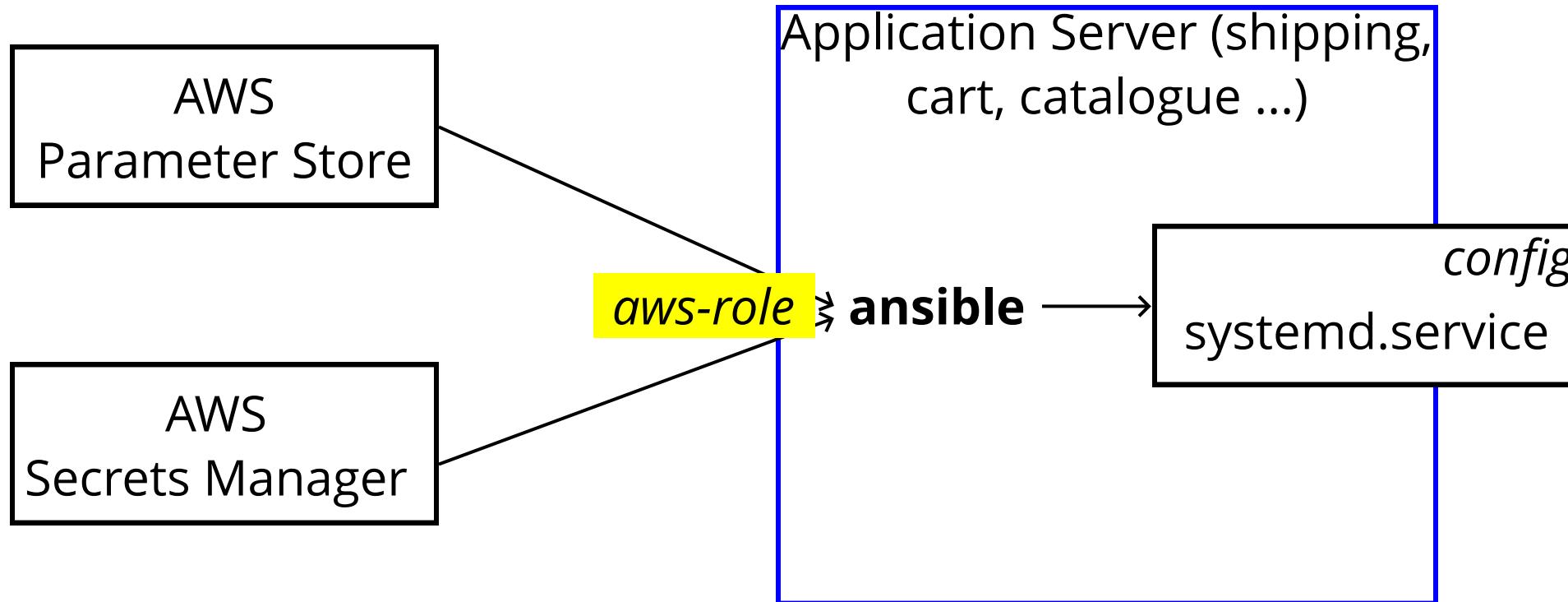
## Parameter Management

- Springboot Config Manager
- Hashicorp Consul
- **AWS Systems Manager - Parameter Store**

## Secret Management

- Hashicorp Vault
- 
- **AWS Secret Manager**

# How Machines will Pull Parameters & Secrets



# AWS SYSTEMS MANAGER

(Parameter Store)



# Monitoring

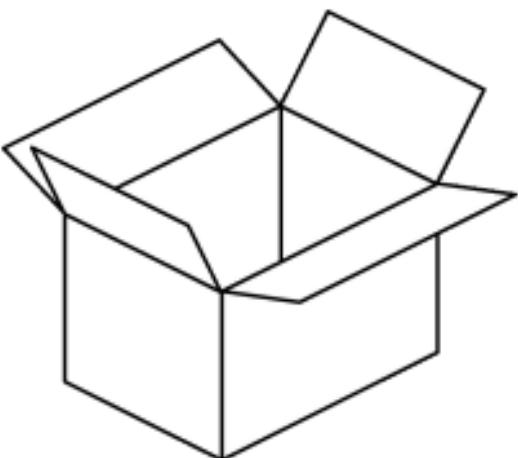
# Four Golden Signals from Google SRE

1. Latency
2. Traffic
3. Errors
4. Saturation

# Prometheus

# **Whitebox Monitoring**

Insights are visible



# **Blackbox Monitoring**

Insights cannot be visible



**Prometheus is Whitebox  
Monitoring**

# Why Monitoring

- Alerts
- Analyze long term trends
- Compare data over time and help you to design / tune the system. Helps you to rightsize the infra.
- Project Dashboards
- Retrospective Analysis
- What is broken and when and why it is broken.

# What Prometheus Does

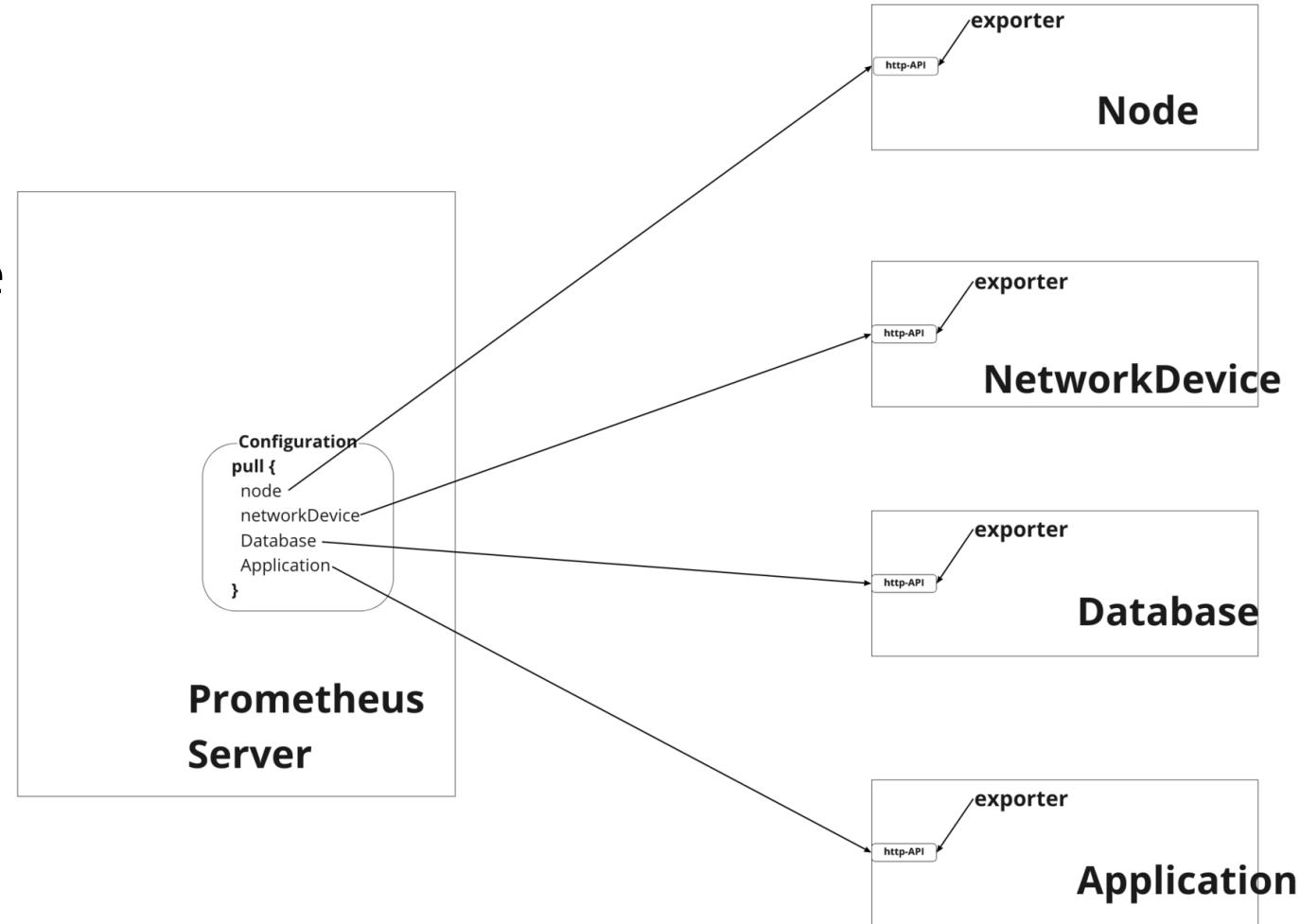
- Metric Collection & Store in TSDB.
- Querying
- Alerting
- Graphing / Trends

# What Prometheus Does no do

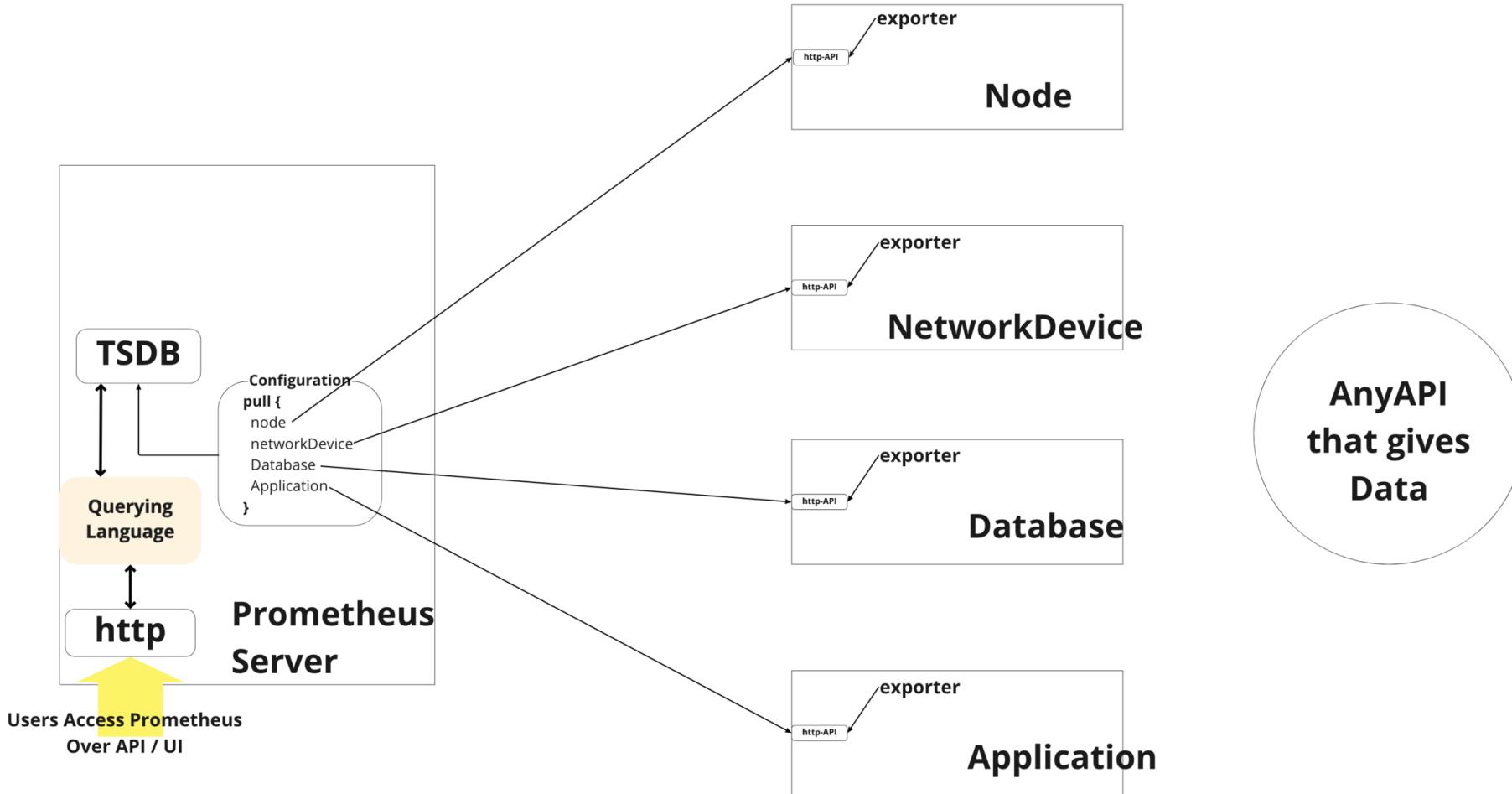
- Raw log / event log (ELK/Splunk)
- Tracing / Data sampling (Instana / Jaeger)
- Does not do ML & Detection
- Long term storage, Max of 15 days (VictoriaMetrics, Thanos)
- Doesn't support Scaling
- Doesn't have Auth Mechanism either for users or even for nodes.

# Prometheus pulls metrics

- Pulls metrics from clients in frequent amount of time as defined in scrape config
- Here client just offers the data over API, Prometheus just collects them.
- Pull enhances to an auto discovery of clients as well and don't require any extra configuration on client side.
- Pull can be made by multiple servers.



# Prometheus Architecture



# Installation

(sudo labauto prometheus-server)

# Metrics are Vectors

# Functions

# Record Rules

```
1 groups:
2 - name: custom
3 rules:
4 - record: node_memory_used_percent
5 expr: ceil(100 - (100 * node_memory_MemAvailable_bytes /
node_memory_MemTotal_bytes))
```

# Alert Rules

```
1 groups:
2 - name: Alerts
3 rules:
4 - alert: InstanceDown
5 expr: up == 0
6 for: 1m
7 labels:
8 severity: critical
9 annotations:
10 summary: "Instance Down - [{{ $labels.instance}}]"
11 description: "Instance Down - [{{ $labels.instance}}]"
```

# Alert Manager

# Service Discovery



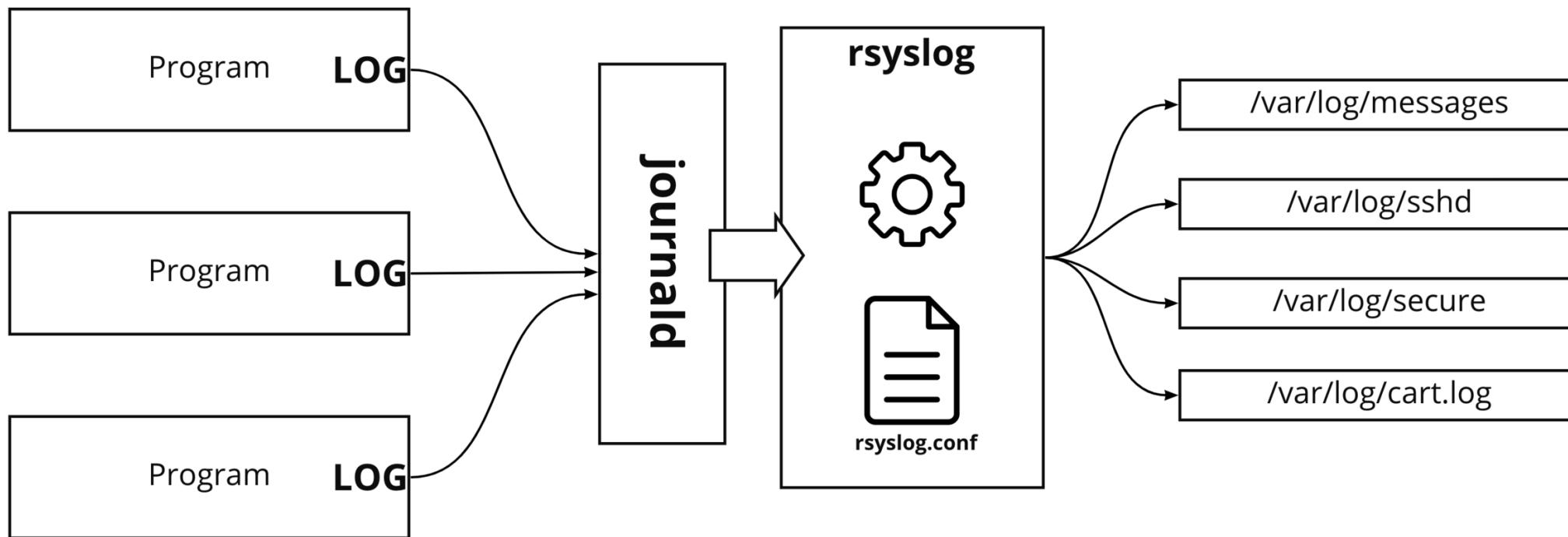
# Logs

- Application Behaviour
- Transactional logs

**Logs writing to Common  
Log file**

# **RSYSLOG in OS**

# How logs ships from program to log files



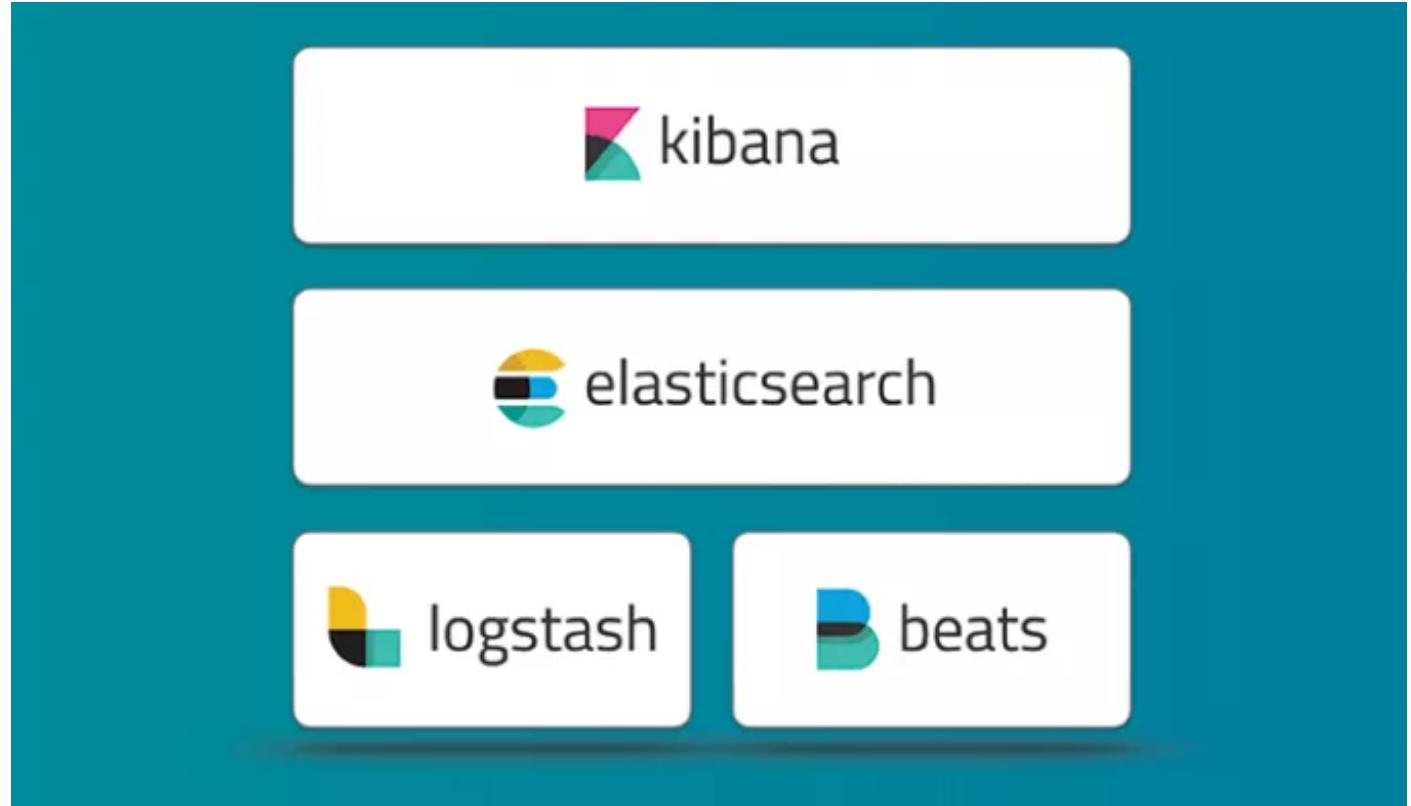
# How logs ships from program to log files

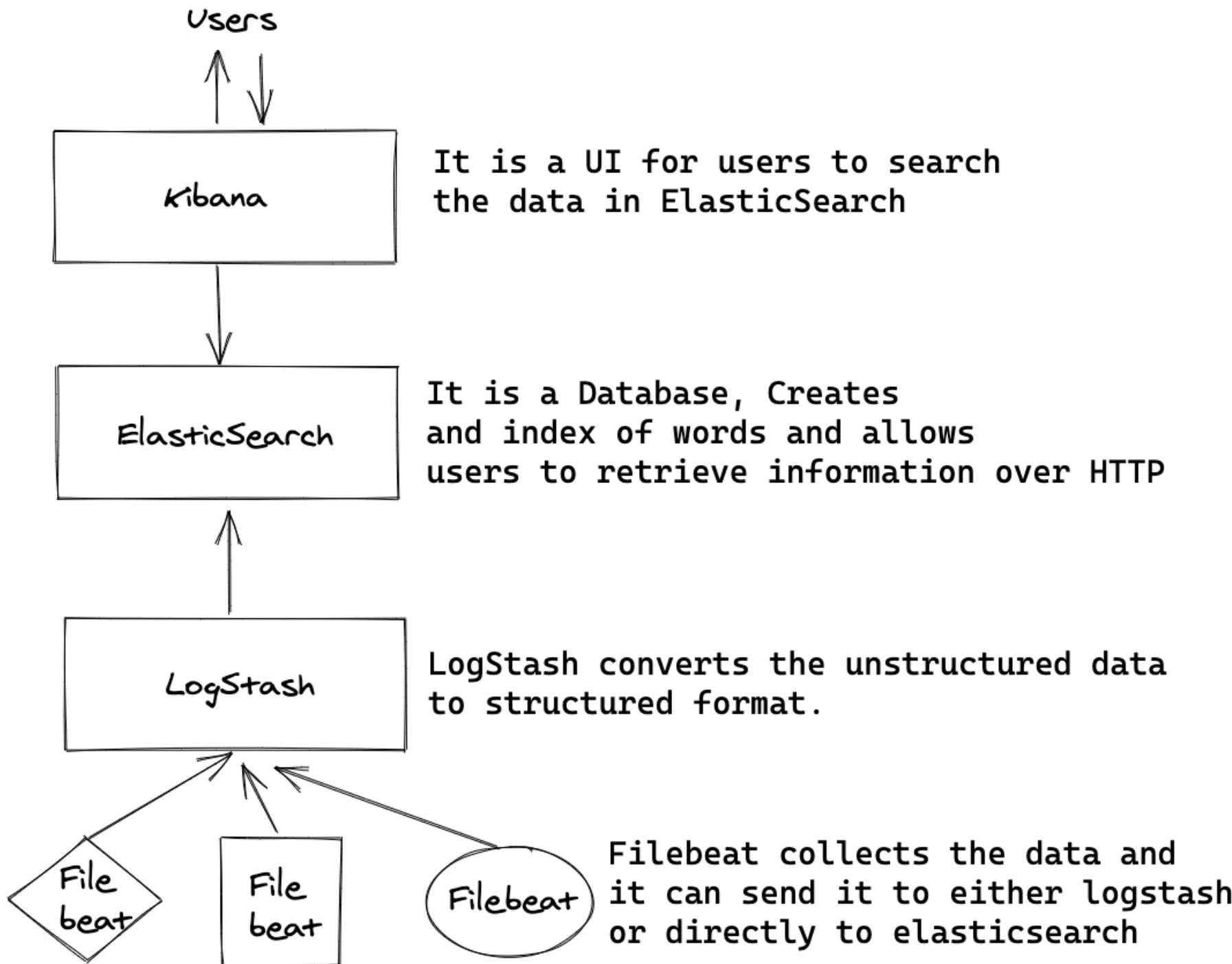
```
[root@cart ~]# cat /etc/systemd/system/cart.service
[Unit]
Description = Cart Service
[Service]
User=roboshop
Environment=REDIS_HOST=redis-dev.roboshop.internal
Environment=CATALOGUE_HOST=catalogue-dev.roboshop.internal
ExecStart=/bin/node /home/roboshop/cart/server.js
SyslogIdentifier=cart

[Install]
WantedBy=multi-user.target
```

```
[root@cart ~]# cat /etc/rsyslog.d/roboshop.conf
template(name="OnlyMsg" type="string" string="%msg:::drop-last-1f%\n")
if($programname == 'cart') then {
action(type="omfile" file="/var/log/cart.log" template="OnlyMsg")
& stop
}
```

# ELK





Doc1

The Linux is good

Doc2

The Linux OS is good

Doc3

DevOps is bad

Doc4

DevOps is high paid

Doc5

Linux is less paid

the 1,2  
Linux 1,2,5  
is 1,2,3,4,5  
good 1,2  
OS 2,  
DevOps 3,4  
bad 3  
high 4  
paid 4,5  
less 5  
this

1 The Linux is good  
2 The Linux OS is good  
3 DevOps is bad  
4 DevOps is high paid  
5 Linux is less paid

# Logs Type

Structured  
Data

```
{
 "level": "INFO",
 "message": "Hello World",
 "date": "2020-10-05"
}
```

Unstructured  
Data

```
2020-10-05 - INFO - Hello World
```

# Structured

- Less operations cost.
- Logstash can be avoided, Means we get fast pushing.
- Adding new fields in the log avoids updating or restarting services.

# Unstructured

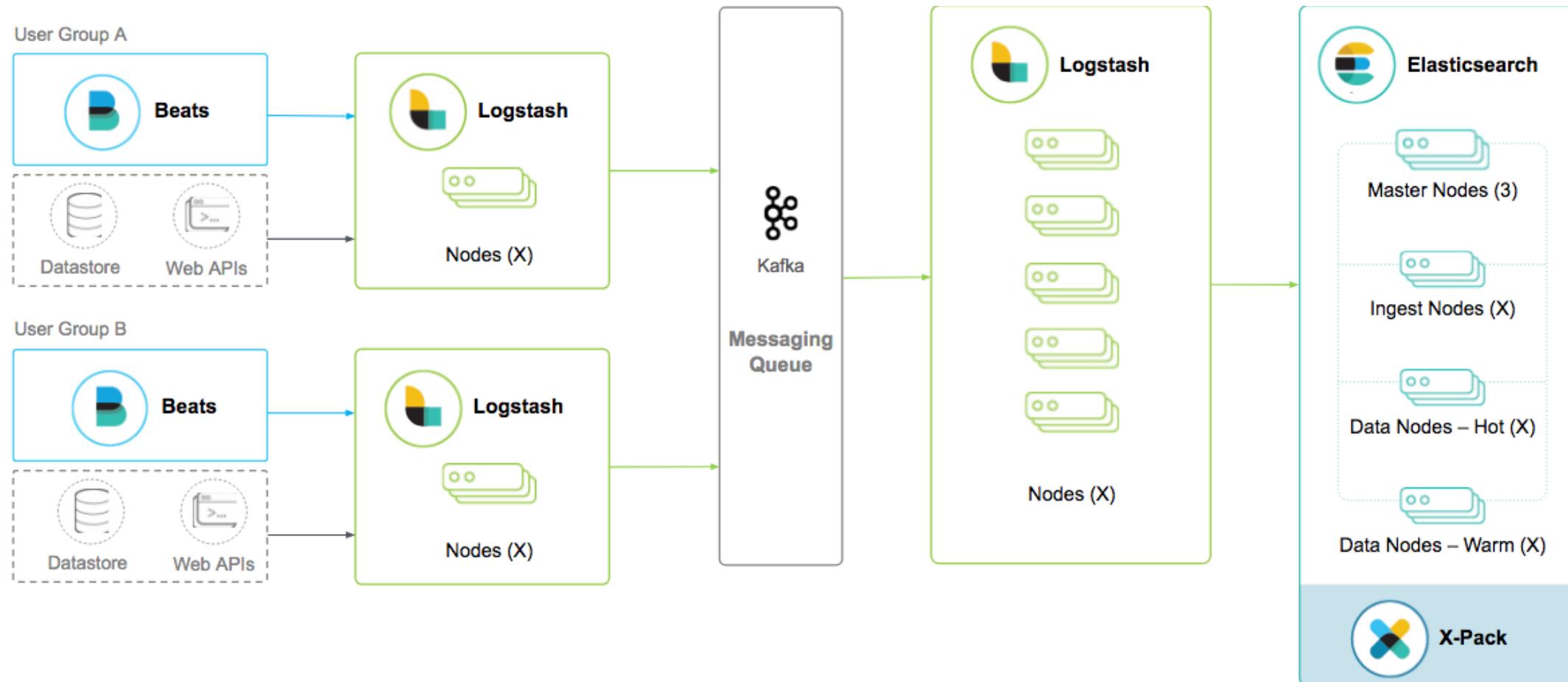
- Operations is harder.
- Logstash crashes is a very common issue, because of its greediness towards memory.
- Logstash is too slow to parse.
- Change in log pattern causes the failure of log parsing

# Installation

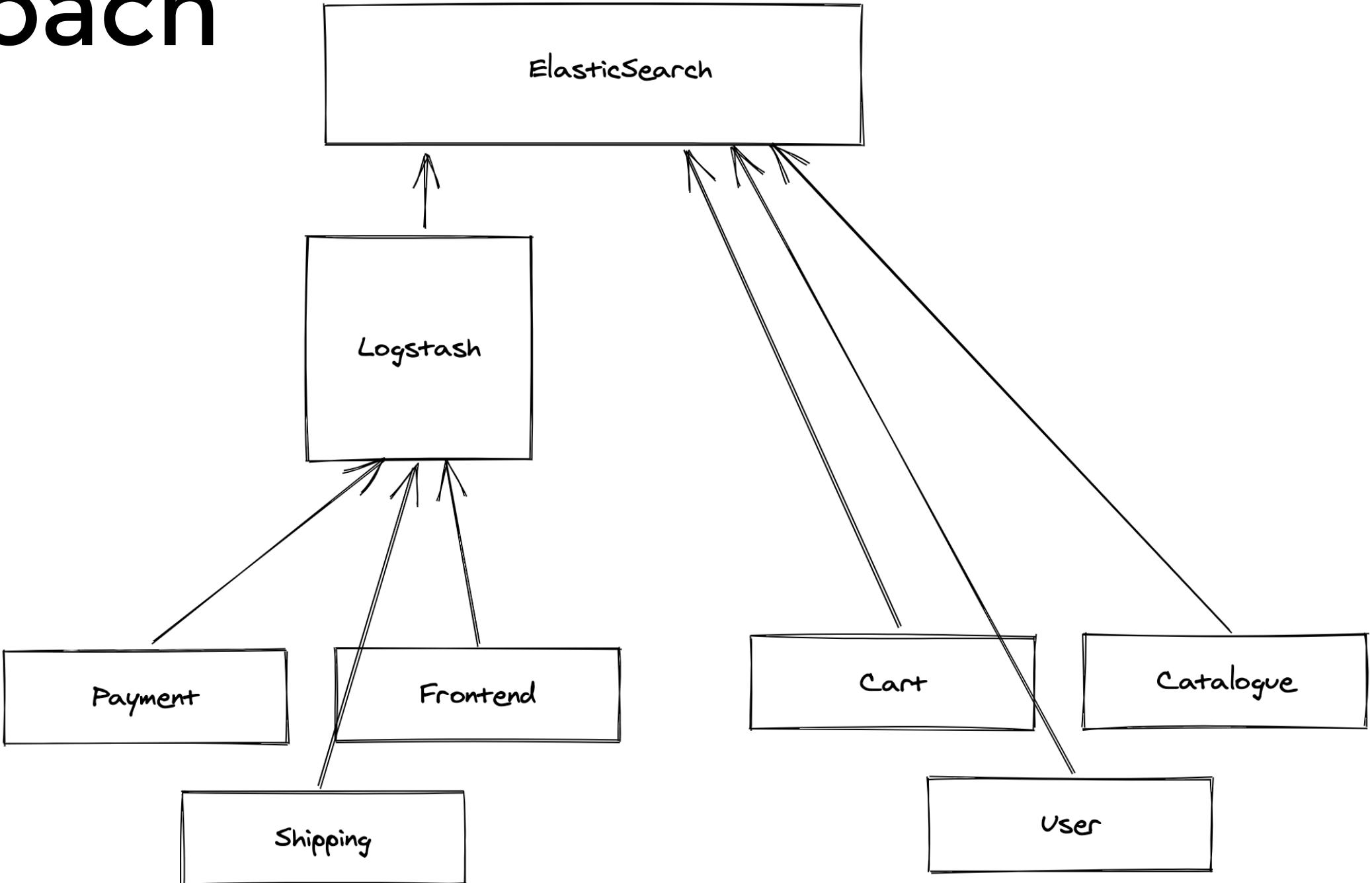
All in One node

All in Multiple node

# Production Grade Cluster



# Approach



# Jenkins

# Jenkins Used for?

Automation

CI & CD

# FreeStyle Jobs

1. **General:** Mostly the high level options on the Job. For example, if need some input provided to the job then **Parameters** have.
2. **SourceCode Management:** Here we provide the git repos to clone and it can be used in later stages of the Job.
3. **Build Triggers:** Different criteria to run jobs automatically in different scenarios.
4. **Build Environment:** Just a step before the actual build steps(Automation Steps) helps in setting up the environment like variables or some secrets to run your build steps.
5. **Build:** Actual steps that we need to run.
6. **Post-build Actions:** After the build if we need to run some steps based on its status like success or failure. For example sending emails about the status of the Job or any such kind of requirements can be achieved.

# Jobs in Jenkins

- **Freestyle Jobs:** Used for dealing with Jenkins over the UI. The problems with this approach are
  1. Track the changes.
  2. Changes are Manual or from UI.
  3. It also has technical limitation on jobs to run based on conditions, loops and kind of scripting approaches.
  4. A lot of plugins are not compatible with FreeStyle Jobs.
- **Pipeline Jobs:** Pipeline is a code, Hence need not to be managed from UI. Advantages here are.
  1. Code changes can be tracked by Git repos.
  2. Changes are from Code.
  3. Pipeline jobs supports loops, conditions & even other coding approaches
- This pipeline jobs helps you to achieve **GitOps**

# Pipeline Jobs

1. Pipeline Code within the Job. (Manage from UI)
2. **Pipeline Code from Git Repository.**

# Pipeline Jobs are two types

- Scripted Pipeline
- Declarative (DSL) Pipeline
- *Declarative (YAML) Pipelines -> Future*

```
node {
}
```

```
pipeline {
}
```

1. Scripted Pipelines are old ways of doing pipelines.
2. Scripted Pipelines are like v1 pipeline code.
3. It is more imperative.

1. Declarative Pipelines are new ways of doing pipelines.
2. Declarative Pipelines are like the latest pipeline code.
3. It is more declarative.
4. Jenkins is more promoting this one.

# Jenkins Post Conditions

| Condition           | Previous State | Current State              |
|---------------------|----------------|----------------------------|
| <b>always</b>       | N/A            | any                        |
| <b>changed</b>      | any            | any change                 |
| <b>fixed</b>        | failure        | successful                 |
| <b>regression</b>   | successful     | failure, unstable, aborted |
| <b>aborted</b>      | N/A            | aborted                    |
| <b>failure</b>      | N/A            | failed                     |
| <b>success</b>      | N/A            | success                    |
| <b>unstable</b>     | N/A            | unstable                   |
| <b>unsuccessful</b> |                | unsuccessful               |
| <b>cleanup</b>      | N/A            | N/A                        |

# Jenkins Pipeline Problems

1. Every time I need to go to UI and create a Job each and every time.  
Creation of Job is manual.
2. Developers are going to generally use multiple branches for the development and single pipeline jobs not a quite fit for this requirement.
3. Every job out there might use some common steps, Now if tomorrow we need to change some config that applies for each and every job, Making changes again on each and every job is not a good idea. So we need to think of keeping code DRY.

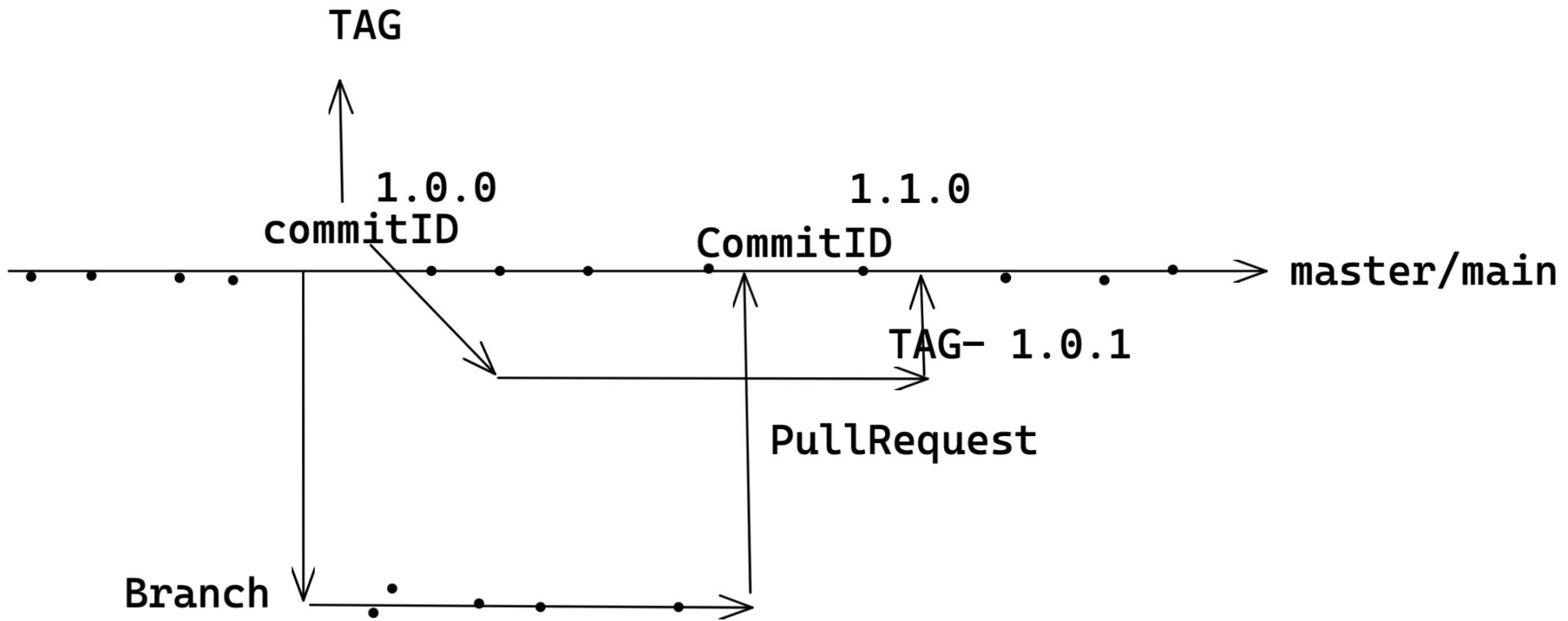
# Create Jobs Automated

1. Jenkins CLI - Jenkins offers command line and a job can be created based on that.
2. Ansible Module - Jenkins can be managed by Ansible, So we can create a playbook to deal with it.
3. Jenkins Job DSL - Jenkins native DSL code to create a job. (Problems with multi branch pipeline)
4. *There are some other ways as well*

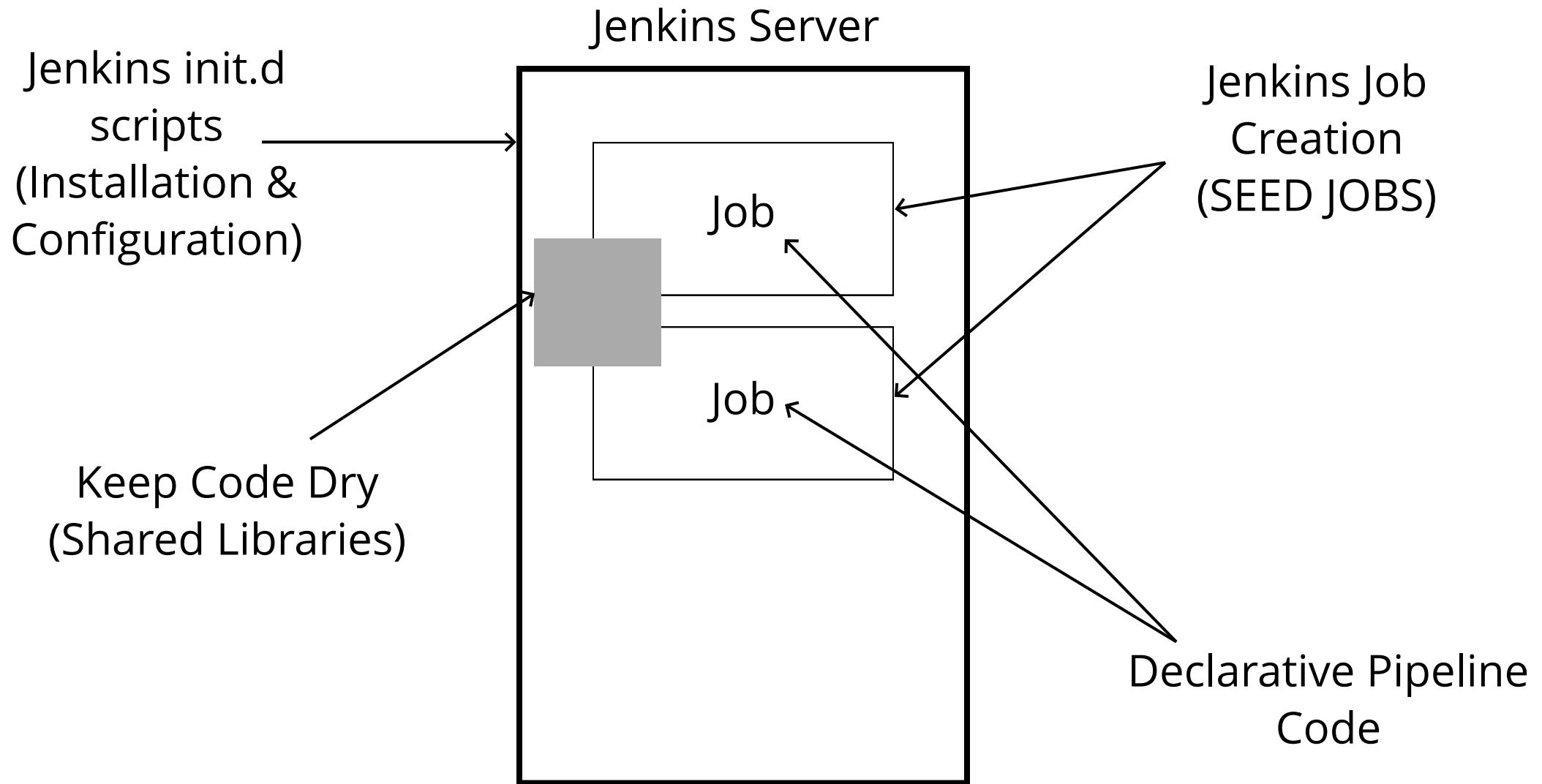
# Multi Branch Pipeline Jobs

Usually, developers may not work on a single branch, they might work with multiple branches. Hence using single branch pipeline jobs is not enough for the development requirements, Hence we need multi branch pipelines where any branch is created without any effort of Developer or DevOps the jobs will be created automatically with multi branch pipelines. Each branch is going to be a separate pipeline in multi-branch pipeline.

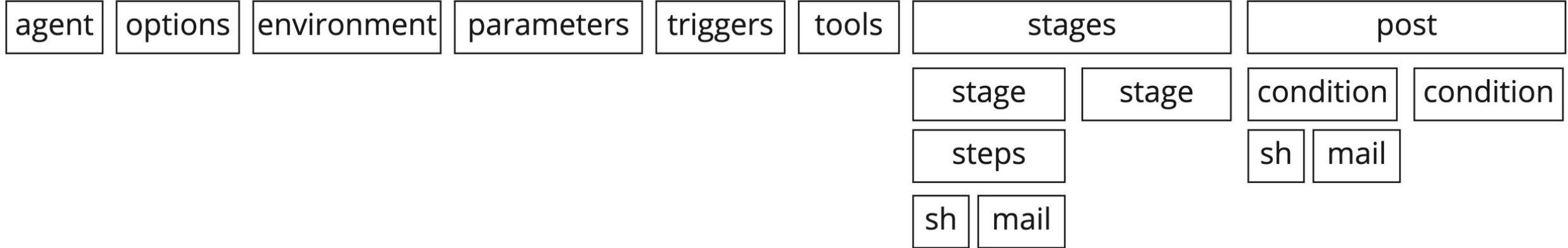
Along with this usually, the software is released with some version number and with git it is going to be tags, When any tags are created in git repos, these multi branch pipelines are going to create a job for that tag and release the software accordingly.



# Jenkins as a Code



# pipeline



# Jenkins Architecture

Jenkins  
Standalone  
Box

1. Jenkins is a standalone architecture. Meaning it cannot run as a cluster. So if that node crashes & leads to down of running all the jobs.
2. To avoid this we kind of decouple the Compute part of Jenkins with Management (Job creation/ Running jobs) of Jenkins.
3. Since compute run on separate machines, Jenkins Jobs can be re-created much faster if we follow some standards.
4. But Jenkins also will have some configuration need to be changed as part of recreating Jenkins server and can be done automated using Groovy init.d scripts.

# Jenkins Agents

- It is used to decouple the compute part from the Jenkins server, So in order to specify on which node we want to run and that can be mentioned in the pipeline using the **agent** section.
- The categorization of Compute Nodes can be flexible.
- These categories can be based on **Environment (DEV,TEST,PROD)**
- These categories can be based on **Tool specific (Ansible, Terraform, Docker)**
- These categories can be based on **Development based ( NodeJS, Java, Golang )**
- These categories can be based on **Cloud (AWS, AZURE)**
- Compute Node automation can be always done with any kind of tool like Ansible.
- Jenkins executers are usually taken based on the need of the parallel jobs.

**Note:** Since we are on lab, keeping cost in mind we may use single node going further.



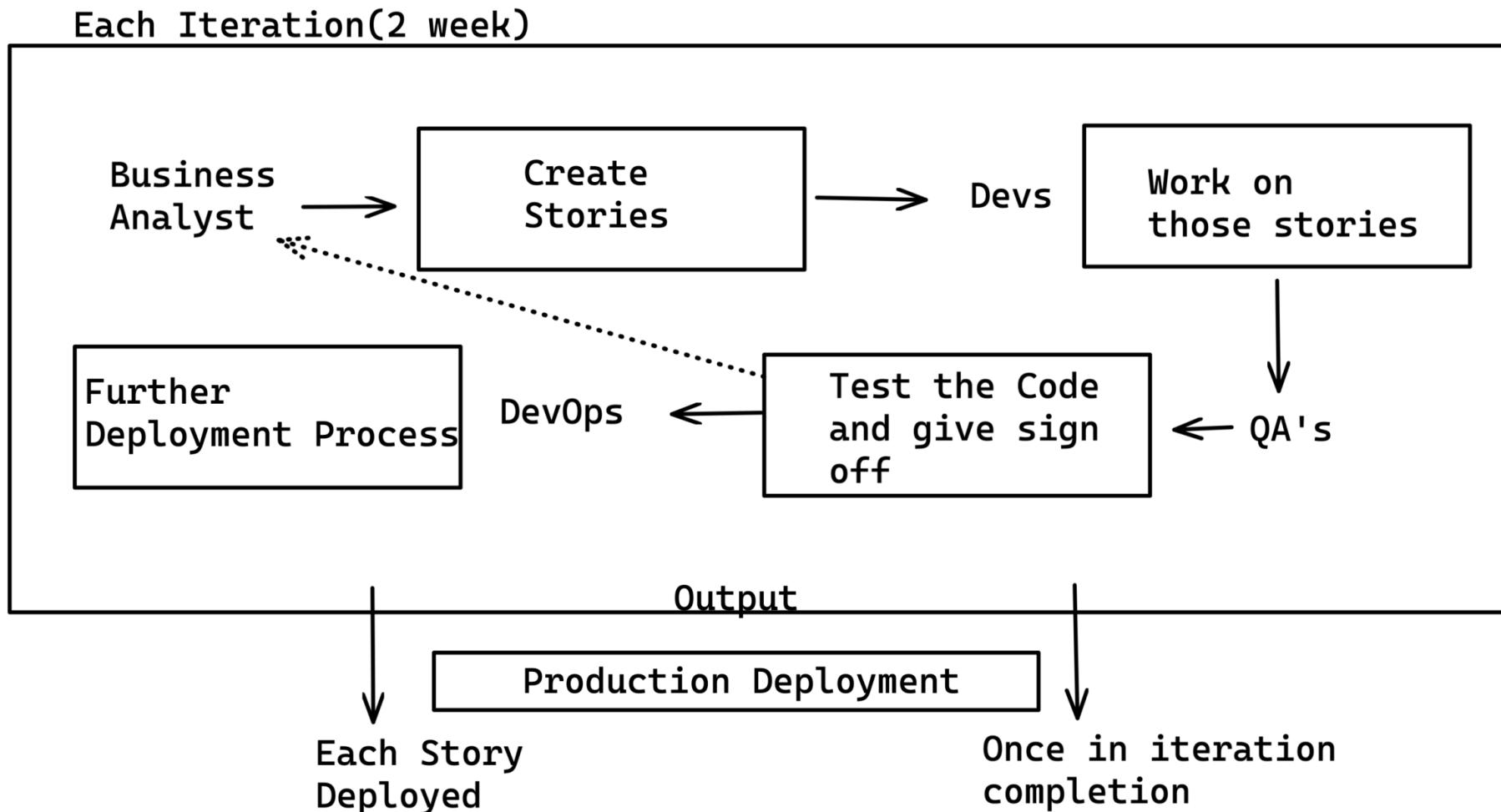
# Jenkins Agents

Add node from  
Jenkins Server

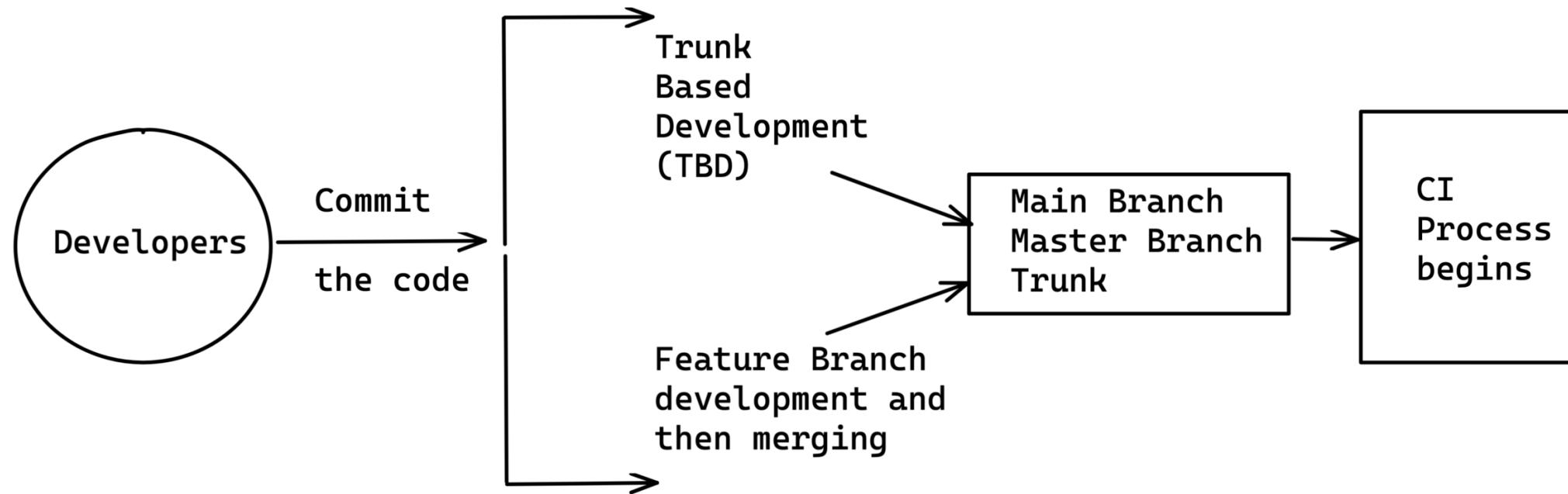
Node reaches  
Jenkins server  
and add itself

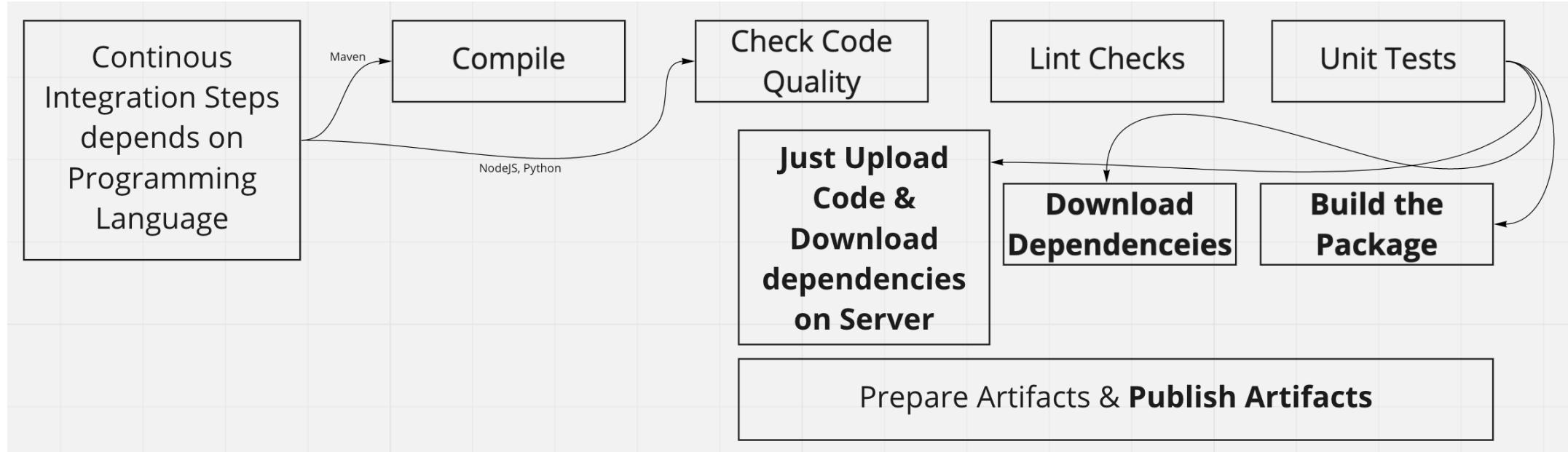
# Continuous Integration

# Work flow in Agile



# How Developers Make Changes





- Compile based on programming language we use that tools, Ex: Java can use Ant, Maven, Gradle.
- Code Quality Happens from Tools like SonarQube.
- Lint Checks can be done by language native tools.
- Unit tests can also be done by 3rd party & native tools of language.
- Next is to prepare the artifacts, Depends on Language again we need to upload to artifact manager accordingly.

# Version Strategy of Different Products

Jenkins

Ansible

Docker

MajorVersion.WekNumber

Using GitTags

Using GitTags

Using GitTags

Decides a custom Number

Manual Git Tag and Pipeline will detect the tag and makes a artifact.

Get Number from CI system

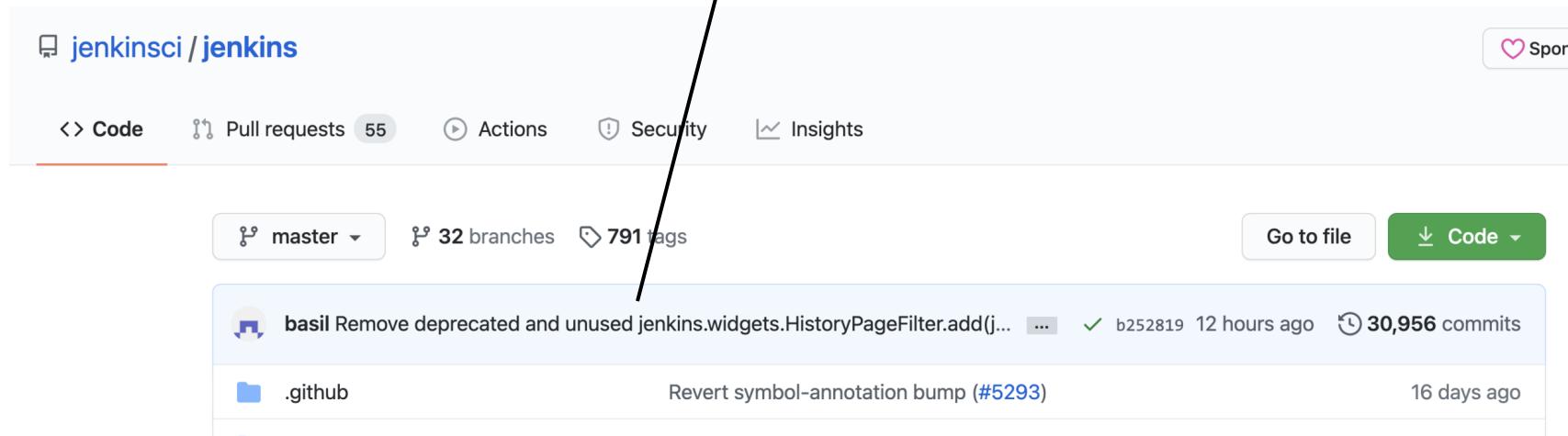
Pipeline uses number input & Makes Artifact using that number

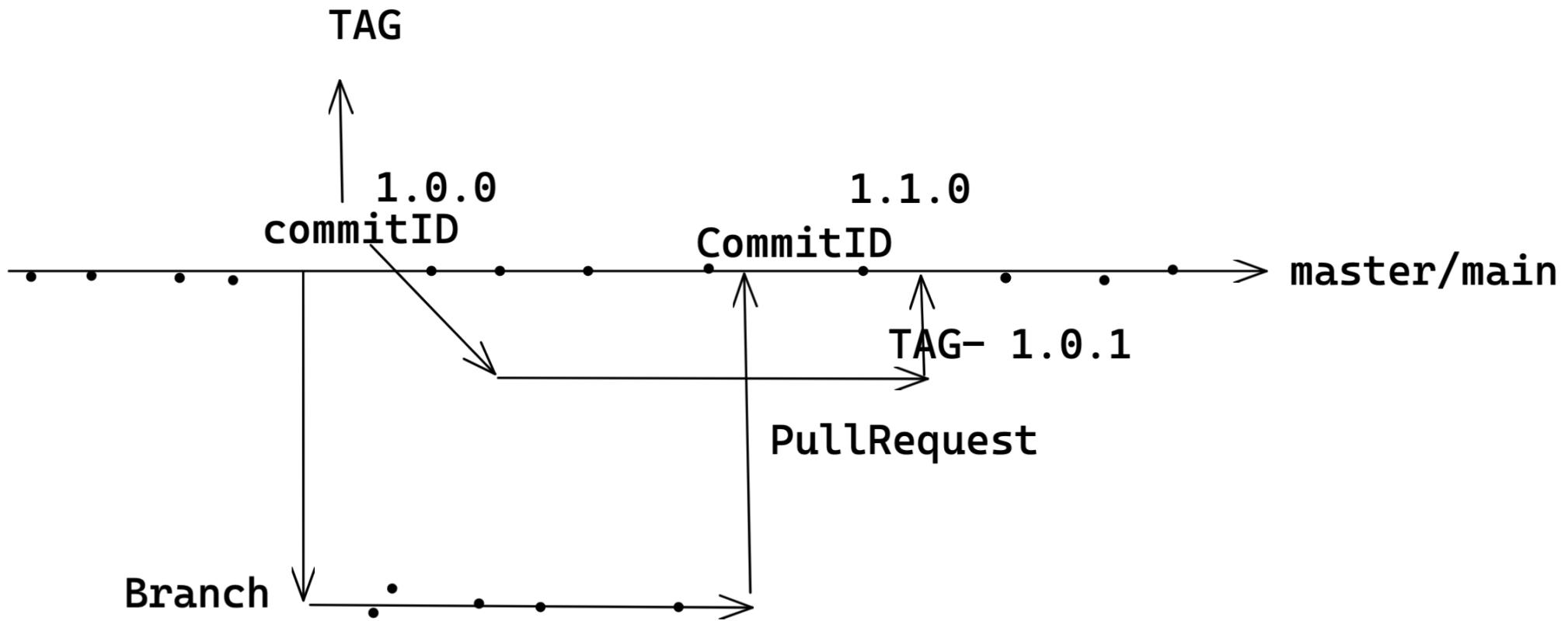
Sprint Number

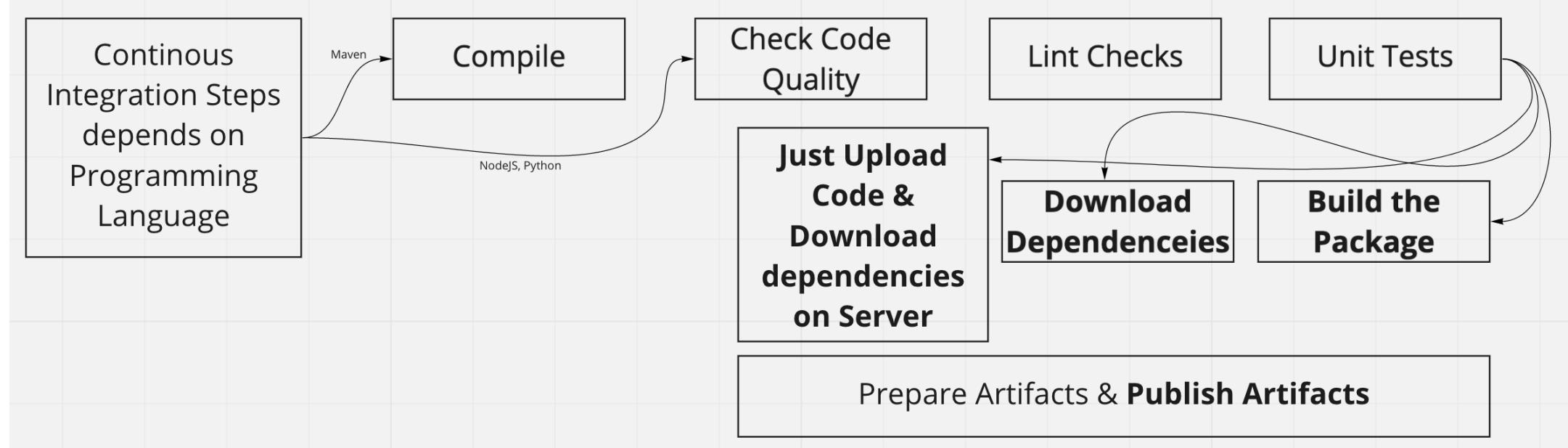
# Version Strategy of RoboShop

Use Git Tags for Releases

Tag  
MajorVersion.SprintNumber.ReleaseNumberINsprint  
Ex:1.1.4



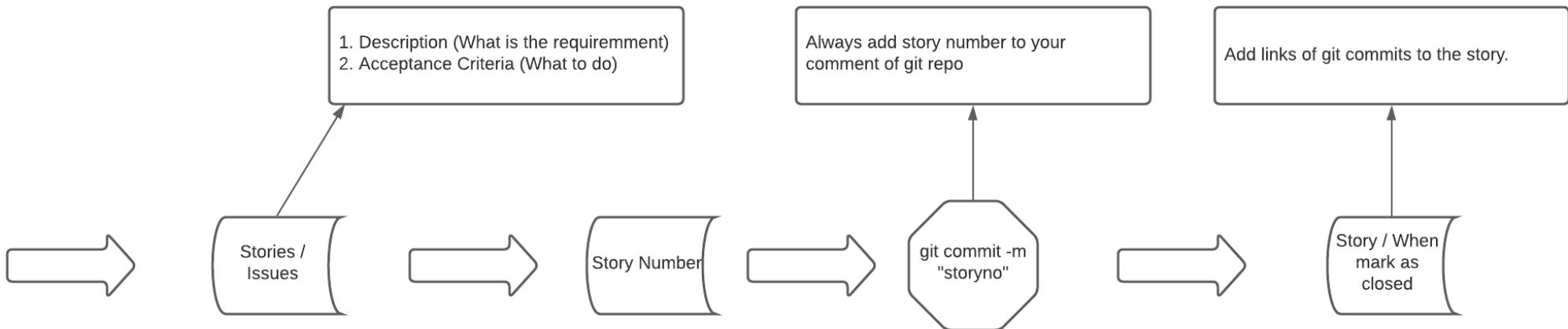




Master / Main : **Compile --> Check Code Quality --> Lint Checks & Unit Tests**

Dev branch : **Compile --> Check Code Quality**

Tag : **Compile --> Check Code Quality --> Lint Checks & Unit Tests --> Prepare & Publish Artifact**



# Test Cases in CI

- **Unit testing** means testing individual modules of an application in isolation (without any interaction with dependencies) to confirm that the code is doing things right.
- **Integration testing** means checking if different modules are working fine when combined together as a group.
- **Functional testing** means testing a slice of functionality in the system (may interact with dependencies) to confirm that the code is doing the right things.



- **Unit testing Example** – The battery is checked for its life, capacity and other parameters. Sim card is checked for its activation.
- **Integration Testing Example** – Battery and sim card are integrated i.e. assembled in order to start the mobile phone.
- **Functional Testing Example** – The functionality of a mobile phone is checked in terms of its features and battery usage as well as sim card facilities.

# CI Steps for Different Languages

|                    | <b>Compile</b> | <b>Packaging</b> | <b>Download Dependencies (CI)</b> | <b>Download Dependencies (Server)</b> |
|--------------------|----------------|------------------|-----------------------------------|---------------------------------------|
| <b>Java</b>        | yes            | yes              | auto (while compiling)            | no                                    |
| <b>GoLang</b>      | yes            | yes              | yes                               | no                                    |
| <b>NodeJS</b>      | no             | no               | yes                               | no                                    |
| <b>Python/Ruby</b> | no             | no               | no                                | yes                                   |
| <b>PHP</b>         | no             | no               | no                                | yes                                   |

# Infra Types

# VPC

1. VPC is a virtual entity.
2. VPC just determines the range and boundaries.
3. Each VPC can have the same network ranges, However, we will try to not conflict with the same range.
4. VPC network range can be determined by estimating the number of IPs needed.
5. Typical IT companies usually go with larger IP ranges to avoid future operations.

# Network Classes

1. Class A - 0.0.0.0 127.255.255.255
2. Class B - 128.0.0.0 191.255.255.255
3. Class C - 192.0.0.0 223.255.255.255
4. *Class D*
5. *Class E*

- These IPs are all from the internet.
- If we have 10 devices and I cannot take 10 ISP connections, One ISP connection will be used to split the internet from ROUTER(wifi) to devices.
- Also we cannot use the public IP for each and every server.

# Private Network

1. Class A - 10.0.0.0 to 10.255.255.255
2. Class B - 172.16.0.0 to 172.31.255.255
3. Class C - 192.168.0.0 192.168.255.255

# Subnets

1. Subnets are actual network configs But created for Availability Zones.
2. Once if the subnet is created then only we can create any compute resources like EC2.
3. Certain people create subnets for all the availability zones.
4. Certain people go and create subnets only on desired number of availability zones. For HA two or more is needed. So people go and use just two also.
5. Based on the subnets created the compute resources will be scattered to multiple AZs for high availability.
6. There is a hidden risk in here, Which is more AZs you use and more data flow from multiple data centers and ultimately more billing comes for Data Transfer charges.

# Servers reachability

If all our servers are not in public, Means no public Ip address then how do we reach the servers.

- Over VPN in AWS
- Over DirectConnet to Company DC
- Over Bastion Node / some Bastion Softwares





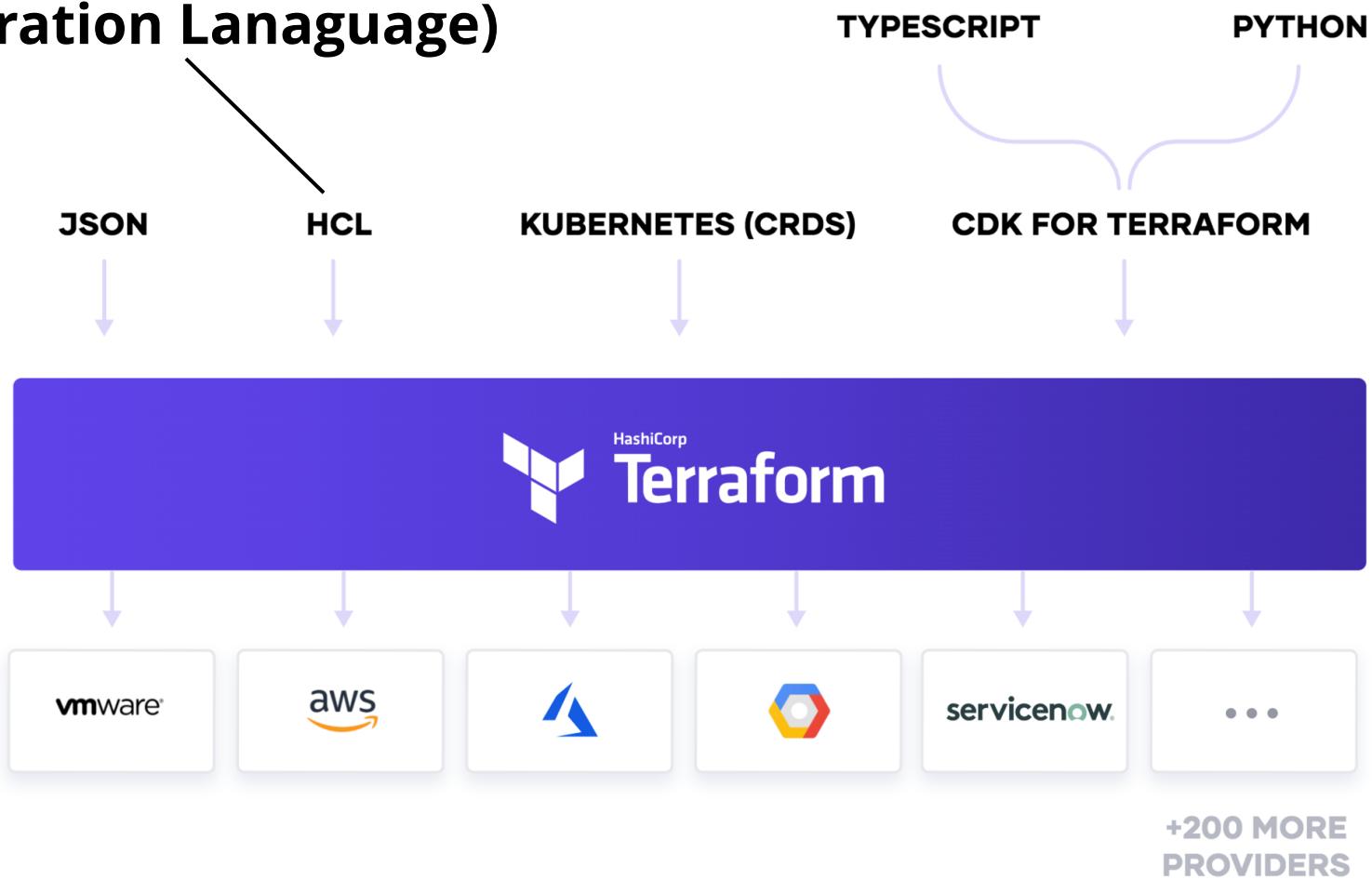
# **Infrastructure as Code (IaC)**

# Why IaC?

- Faster time to production releases
- Improved consistency, less configuration drift
- Faster, more efficient development
- Lower costs and more time developing and fewer operations.

# Terraform

We go with HCL (Hashicorp Configuration Lanaguage)



# HCL v2 (Current)

# How HCL will have the code

```
resource "aws_instance" "web" {
 ami = "ami-a1b2c3d4"
 instance_type = "t2.micro"
}
```

- Code is in blocks, The example you see here is a **resource** block.
- A resource belongs to a **Provider**.
- Terraform has a lot of providers. **AWS** is one of them.

- Everything is a block in terraform HCL.
- Ex: resources, variables, outputs, data, provider, locals, module

# Terraform Files

- All the files of terraform should end with **.tf** or **.tf.json** file extension.
- We can keep multiple files, Files will be loaded in terraform in alphabetical order, but it compiles the list and make its own order.
- Execution order will be smartly picked by terraform, Also gives the flexibility to write your own dependencies (**depends\_on**).

# Terraform Command

- Terraform echo system comprises of **init, plan, apply, destroy**.
  - **Destroy** is optional unless you want to destroy the resources created.
1. **INIT** - This phase downloads all the required provider plugins and also initializes the state file if it is remote.
  2. **PLAN** - Plan will show what the terraform can do on your code when you actually apply.
  3. **APPLY** - Create the actual resources.
  4. **DESTROY** - Delete the actual resources which are created.

# Outputs

```
output "instance_ip_addr" {
 value = aws_instance.server.private_ip
}

output "sample" {
 value = "Hello World"
}
```

- Output prints a message on the screen.
- Output block helps in printing the created resource attributes & arguments on the screen.
- Outputs with modules work as a data transmitter.
- You can define multiple output blocks.

# Terraform Datasources

```
data "aws_ami" "example" {
 executable_users = ["self"]
 most_recent = true
 name_regex = "Centos-8-DevOps-Practice"
}

output "AMI_ID" {
 value = data.aws_ami.example.id
}
```

- Data sources are used to refer the data of existing resources in the provider.

# Variables

```
variable "sample" {}

variable "sample1" {
 default = "Hello World"
}
```

```
output "sample" {
 value = var.sample
}
```

# Variable - Data Types

```
String Data type
variable "sample1" {
 default = "Hello World"
}

Number data type
variable "sample2" {
 default = 100
}

Boolean Data type
variable "sample3" {
 default = true
}
```

Terraform supports data types and those are

1. Strings
2. Numbers
3. Booleans

Strings data should be quoted in double-quotes, But whereas numbers and booleans need not to be.

Terraform only supports double quotes not single quotes

# Variables Types

Default Variable Type

```
1 variable "sample" {
2 default = "Hello"
3 }
```

List Variable Type

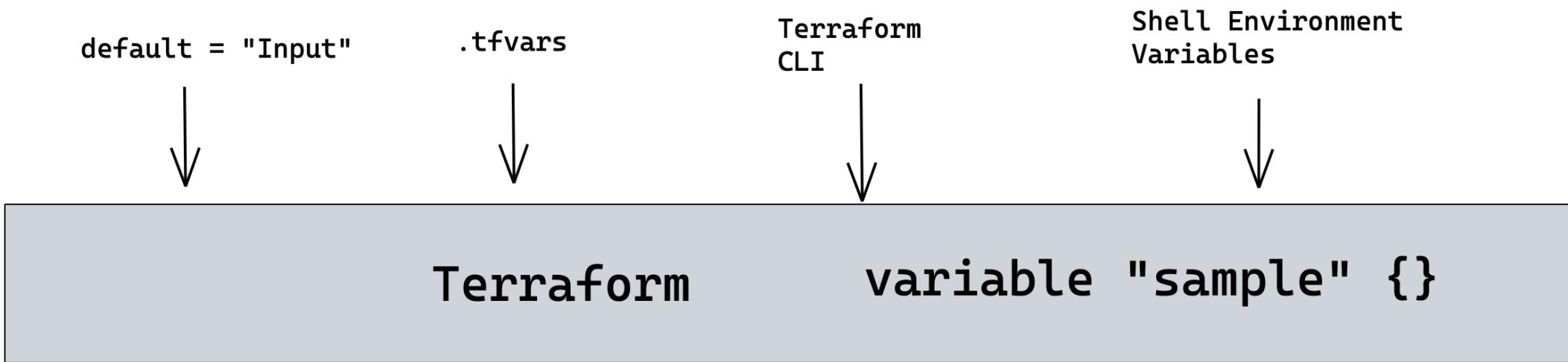
```
1 variable "sample" {
2 default = [
3 "Hello",
4 1000,
5 true,
6 "World"
7]
8 }
```

Map Variable Type

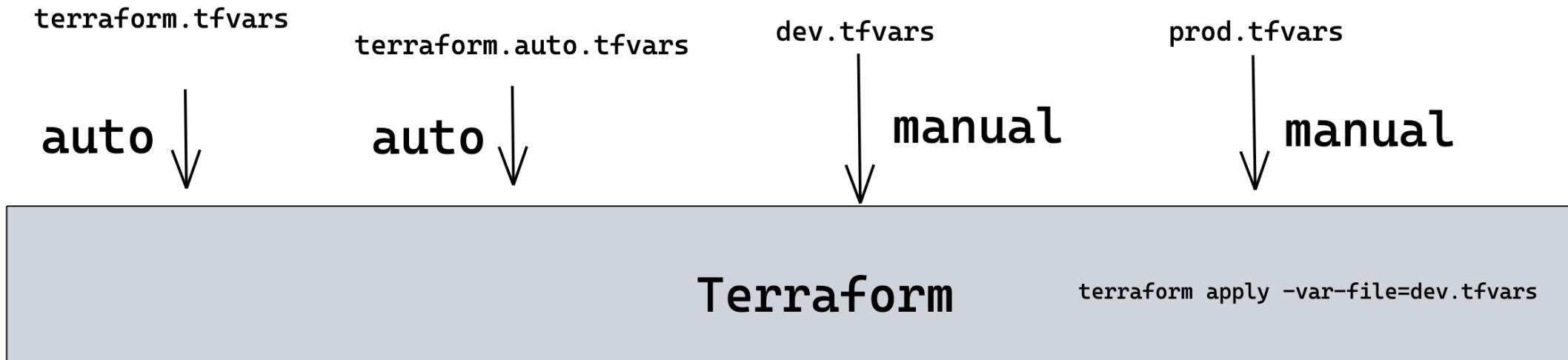
```
1 variable "sample" {
2 default = {
3 string = "Hello",
4 number = 100,
5 boolean = true
6 }
7 }
```

Terraform supports different data types in a single list or map variable, Need not to be the same data type.

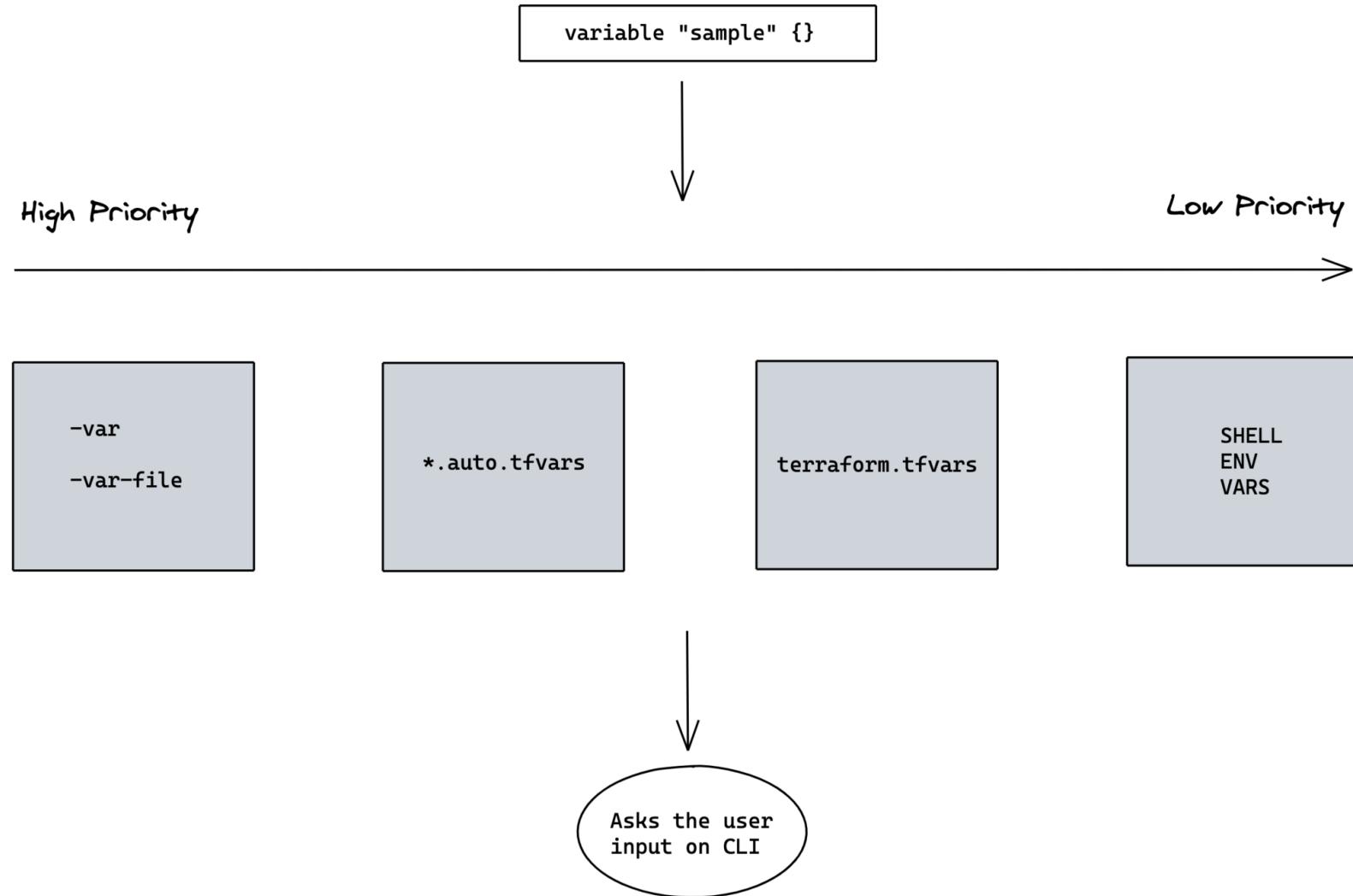
# Variables Inputs



# Variables TFVARS



# Variables Precedence



```
terraform {
 required_providers {
 prod = {
 source = "hashicorp/aws"
 version = "1.0"
 }
 dev = {
 source = "hashicorp/aws"
 version = "2.0"
 }
 }
 provider "aws" {}
 provider "azurerm" {}
```

# Provider

- A sample provider is what you see on here.
- You can deal with multiple providers at the same time.
- You can specify a particular version of the provider or the latest will be used if not provided.
- You can also use two providers declarations needed for same provider sometimes, Like AWS dealing with multiple accounts or multiple regions. That is possible with **alias**

# Resources

```
resource "aws_instance" "sample" {
 ami = "ami-052ed3344670027b3"
 instance_type = "t2.micro"
}

output "public_ip" {
 value = aws_instance.sample.public_ip
}
```

- Resources have **arguments** as keys. **ami & instance\_type** arguments
- Resource block will have two names, One is provider resource name and other is your local resource name. **aws\_instance** is provider resource name & **sample** is local resource name.
- Resource will be referred as <provider\_resource\_name>.<local\_resource\_name>
- Resource expose the information after creating it , those called as **attributes**. **public\_ip** is an attribute to resource of **aws\_instance**

# Terraform Code Indentation

```
resource "aws_instance" "sample" {
 ami = "ami-052ed3344670027b3"
 instance_type = "t2.micro"
 tags = {
 Name = "Sample"
 }
}

output "public_ip" {
 value = aws_instance.sample.public_ip
}
```

- Two spaces in any new block or even nested block
- Align all your equal signs

# Create Instance with One Security Group

Q. What happens if we run terraform apply multiple times. Does it create multiple resources again and again ?

A: No

Q: How?

A: Terrafrom State file.

Q: What is it?

A: When terraform applies, then it stores the information about the resources it has created in a form of a file and it is called a **State file**

# Terraform State File

- Terraform by default create a **tfstate** file storing the information of the resources created.
- That file is a very important file, meaning if we loose the file and we lost the track of all the resources created.
- The strategy of saving the terraform state file is a quite important thing.
- Terraform extends it support and offers to store this state information in a database.
- Storing on a database will need to have a database running.
- But cloud providers also extend their support to store the file of Terraform state file, In AWS we can use **s3** bucket.

# Terraform Remote State File

- Simply choosing a remote state of s3 will cause some more problems.
- Assume if multiple people are running the same code updating the same state file and that leads to corrupting the state file.
- We need to ensure that we have a locking mechanism available for remote state files.
- Terraform with AWS extends its support to have a locking mechanism with DynamoDB database integrated with S3 state file.

# Terraform Modules

```
1 module "sample" {
2 source = "./ec2"
3 }
4
5 module "consul" {
6 source =
7 "github.com/hashicorp/example"
8 }
```

- Module is nothing but a group of resources.
- It helps you in keeping your code DRY.
- In Terraform everything is a module and the main code is called as **Root Module**.
- You cannot share the data from one module to another module directly. But can be done only through **Root Module**.

## Share data from one module to another

module.a1

1. Declare  
output

module.a2

variable "abc" {}

3. Refer that  
variable in  
resources

module.root

2. Send the module.a1 output to module.a2  
as input

```
module "a2" {
 source = ""
 abc = module.a1.abc
}
```

# Terraform Loops

```
1 resource "aws_instance" "sample" {
2 count = 3
3 ami = var.AMI[count.index]
4 instance_type = var.INSTANCE_TYPE
5 }
```

- Loops in terraform help you to create multiple resources using the same resource block by iterating it.
- Keyword **count** will tell to the resource that howmany times it has to iterate.
- Count offers index and that index can be referred to as **count.index**. Index starts from **0**
- Starting from terraform 0.13 version **count** is available for module also.

# Terraform Conditions

```
1 resource "aws_instance" "sample" {
2 count = condition ? true_val : false_val
3 ami = var.AMI[count.index]
4 instance_type = var.INSTANCE_TYPE
5 }
6 # count = var.CREATE ? 1 : 0
7 # count = var.ENV == "PROD" ? 1 : 0
8
9 variable "CREATE" {
10 default = true
11 }
```

- Conditions is all about picking up the value for the arguments.
- Conditions in terraform are to just determine whether a resource can be created or not using **count** keyword.
- If the **condition** is true then it picks **true\_val** else it picks **false\_val**.
- It is majorly used with count , But can also be used with any argument as well.

# Terraform Functions

- Functions are inbuilt in terraform. We cannot create any functions as user of terraform.
- These bring some common programming functionalities like **Convert to Upper Case, Cut a String, Add some number....**
- Function can be used with arguments in resources and modules also.

# Terraform Variable Validations

```
1 variable "image_id" {
2 type = string
3 description = "The id of the machine image (AMI) to use for the
server."
4
5 validation {
6 condition = length(var.image_id) > 4 && substr(var.image_id,
0, 4) == "ami-"
7 error_message = "The image_id value must be a valid AMI id,
starting with \"ami-\"."
8 }
9 }
```

- To avoid the later code failures as a best coding practice we always validate the inputs provided by user.
- Terraform extends its support to do this in variables using validations with the help of functions.

# Terraform Provisioners

- Terraform provisioners are used to execute certain tasks after the resource creation, For example, Connect to instance and perform some commands, Copy some files to the instance that got created.
- Provisioner is a sub-block in resource.
- Resource attributes can be used inside provisioner and those should be referred as **self.<attribute>**
- Provisioners by default are **create-time-provisioners** and we can also specify **destroy-time-provisioners**.
- We can make multiple provisioners and provisioner types also in the same resource.

# Best Practices on Terraform Code

- Group the resources based on the requirement.
- Modularize the code in to small pieces.
- Modularize will help in enterprise organizations on controlling the access of what resources you are suppose to run.
- Always prefer to keep modules in separate repos. Because it brings a larger advantage in working with multiple environments.

# Project Code Structure

# Terraform In-Built Functions

- Terraform have inbuilt functions to manage and transform the data.
- It has numerous functions available.
- A user cannot define their own function in terraform.

# Terraform Code Project Way



The screenshot shows a dark-themed code editor interface. On the left, there is a sidebar titled 'VPC' containing a tree view with nodes: 'env-dev', 'env-prod', 'module-vpc', 'main.tf', and 'vars.tf'. The 'main.tf' node is currently selected. On the right, the main editor area displays the contents of the 'main.tf' file:

```
1 module "vpc" {
2 source = "./module-vpc"
3 }
```

- This way of code will be used for only project, however for multiple environments.
- Roboshop as a project. Now all the components like cart, catalogue and so on will be running in one place, In other words it is one VPC.
- Module and project code are local repo.

# Terraform Code Platform Way



The image shows a dark-themed code editor interface. On the left, there's a sidebar with a tree view of files:

- VPC
  - env-dev
  - env-prod
- main.tf
- vars.tf

The main editor area shows the content of the `main.tf` file:

```
main.tf > ...
1 module "vpc" {
2 source = "http://github.com/vpc"
3 }
4 |
```

- This way of code will be used for multiple projects, and even for multiple environments.
- Roboshop as a platform or Roboshop can be a part of platform.
- RoboShop as a platform, Now all the components like cart, catalogue and so on will be running on different places, Meaning different VPCs hosting that application.
- In this mode we need modules to be centralized so that multiple projects of a platform can use the same code and keep the code DRY.

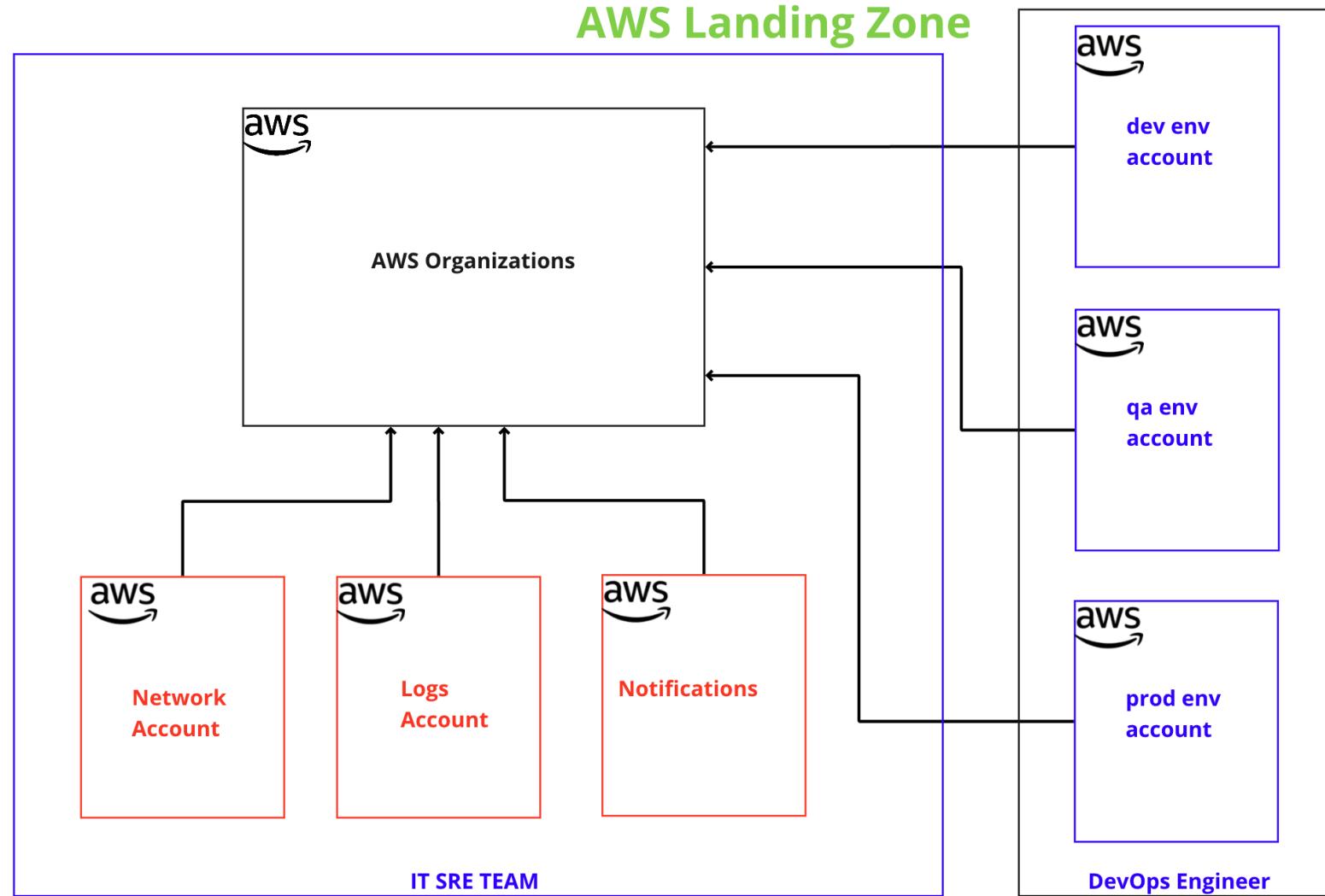
# In LAB, Terraform Code in Platform Way, But use in project way

This way though we start with one project tomorrow if new projects come into picture for our company we can share the code of terraform between these projects and even multiple projects.

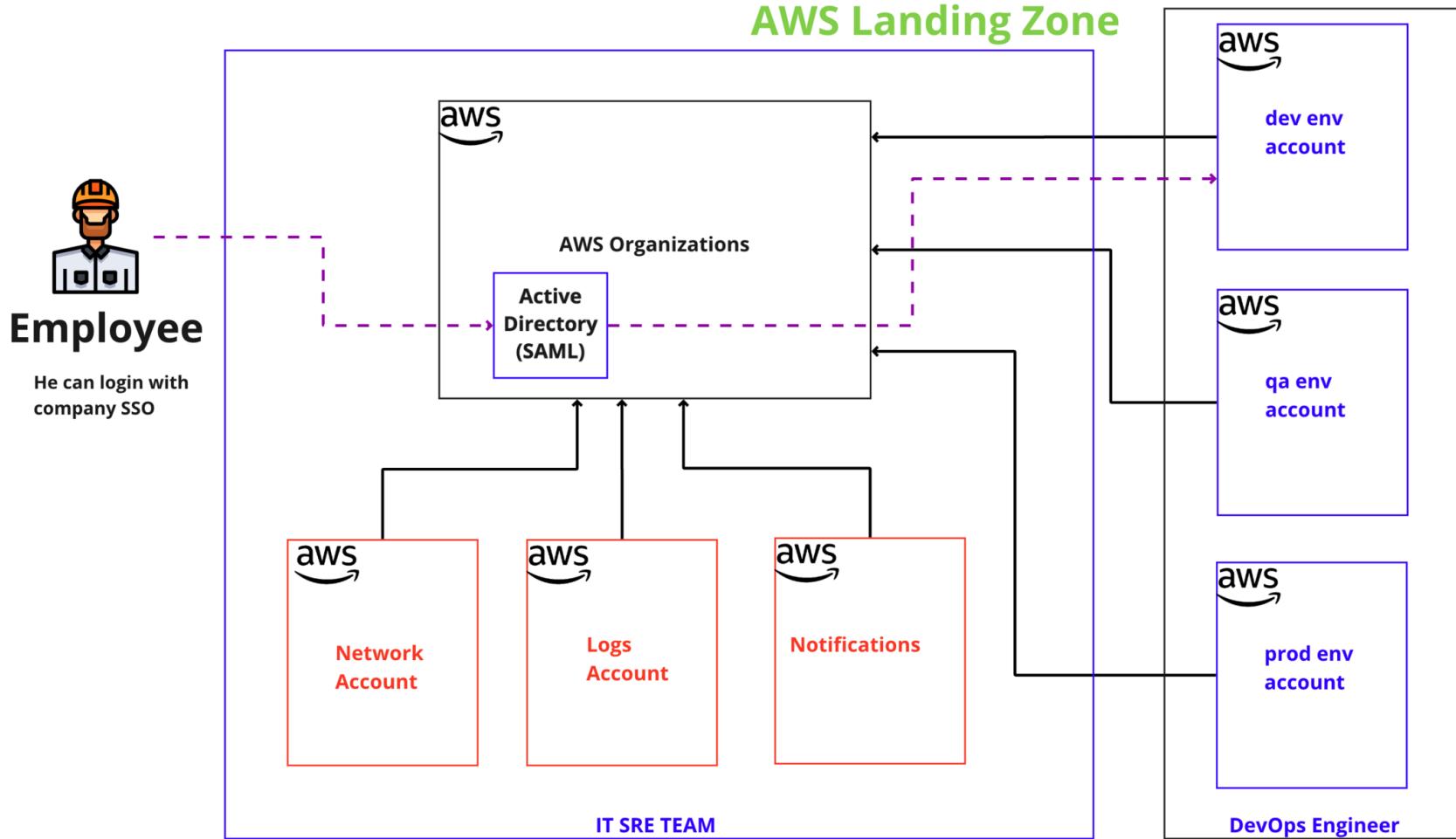


# Projects Landscape

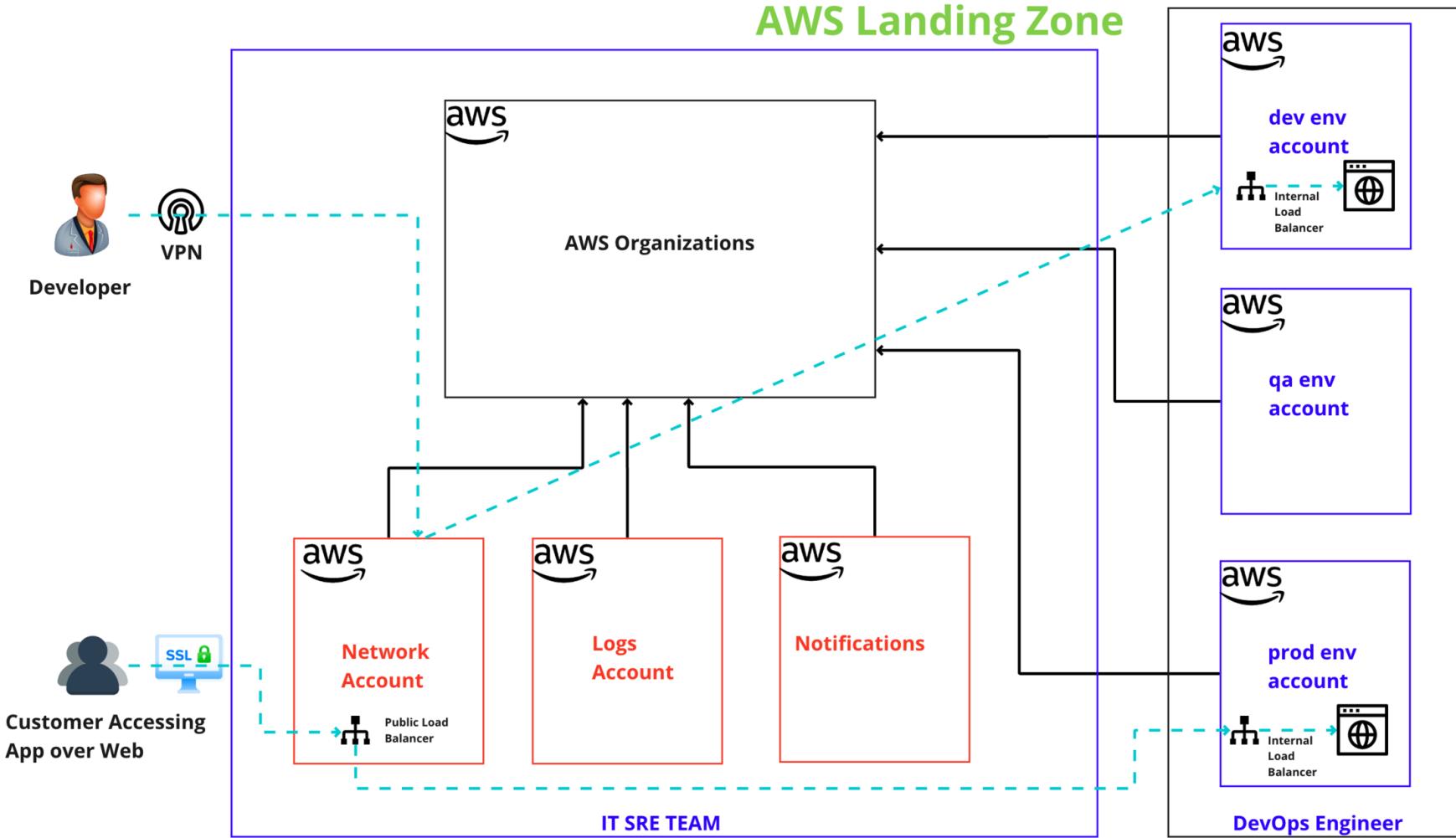
# AWS Landing Zone



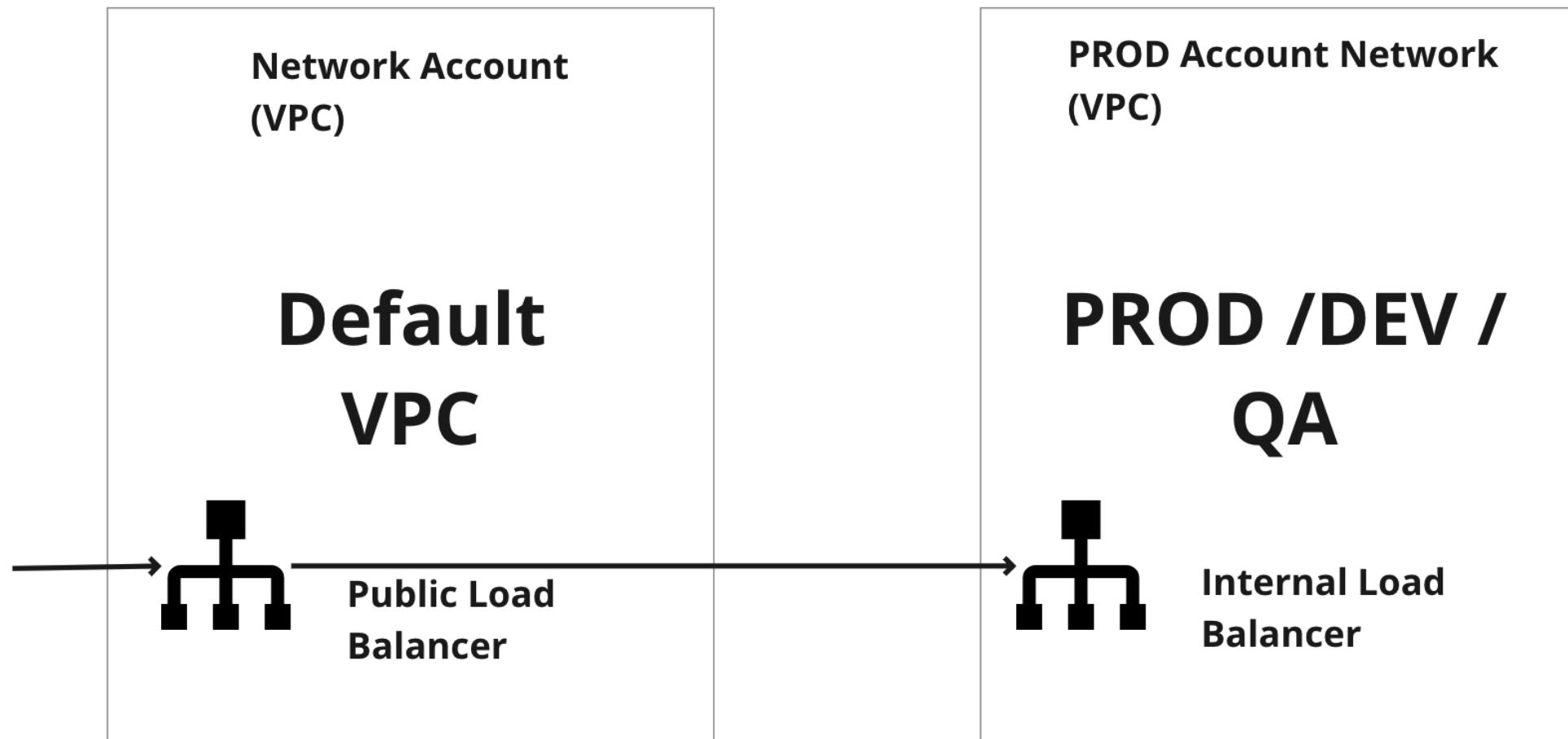
# AWS Landing Zone - Employee Authentication



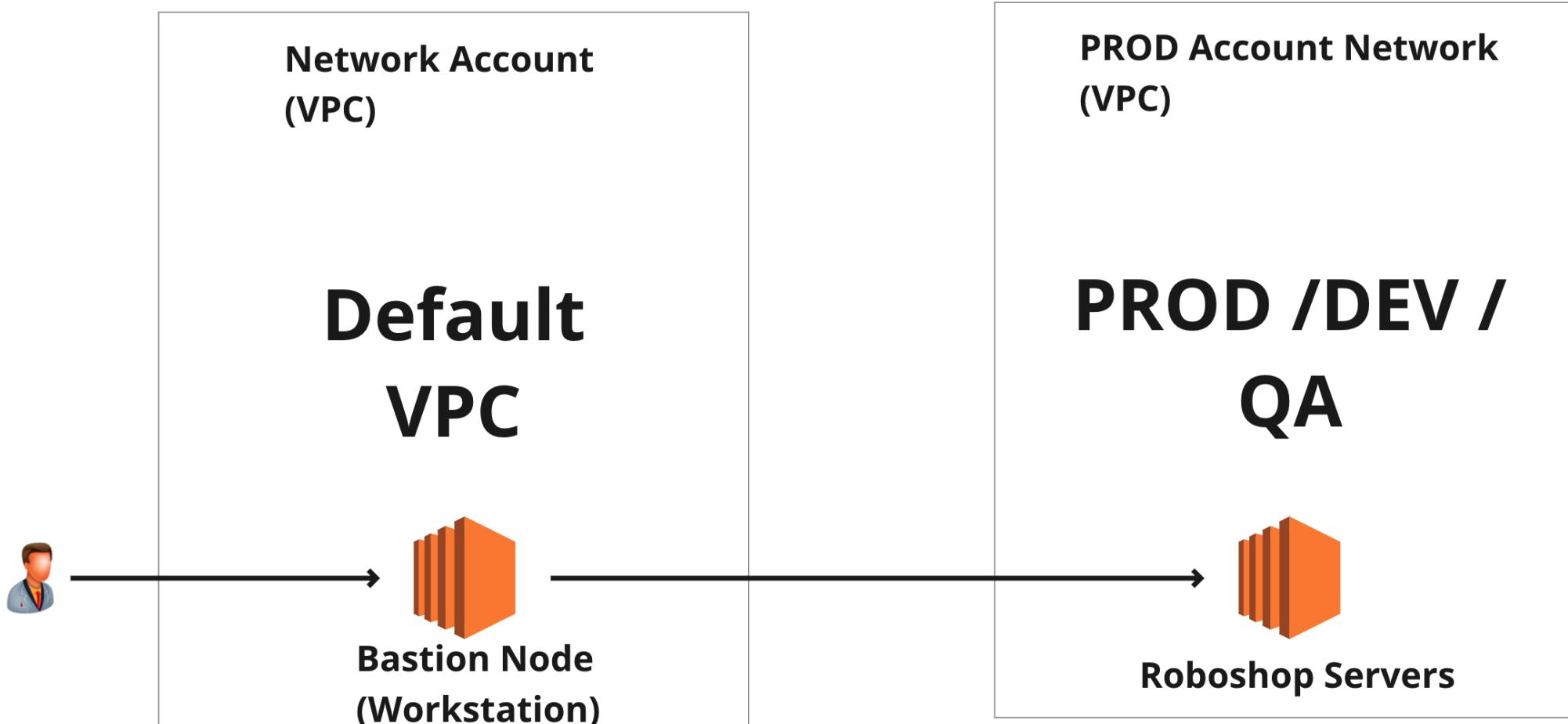
# AWS Landing Zone - Network Flow



# LAB - App Network Flow



# LAB - VPN Network Flow



# Network Sizing

**Q:** How to size the network?

**A:** Based on number of IPs we need.

That means we need to forecast howmany number of machines and IPs come into existence.

**Q:** Can we really forecast exactly howmany are needed ?

**A:** Yes we may do an approximate calulcatation.

**Q:** Is there any anyway we start with not forecasting, Also at the same time we do not hit the limitations?

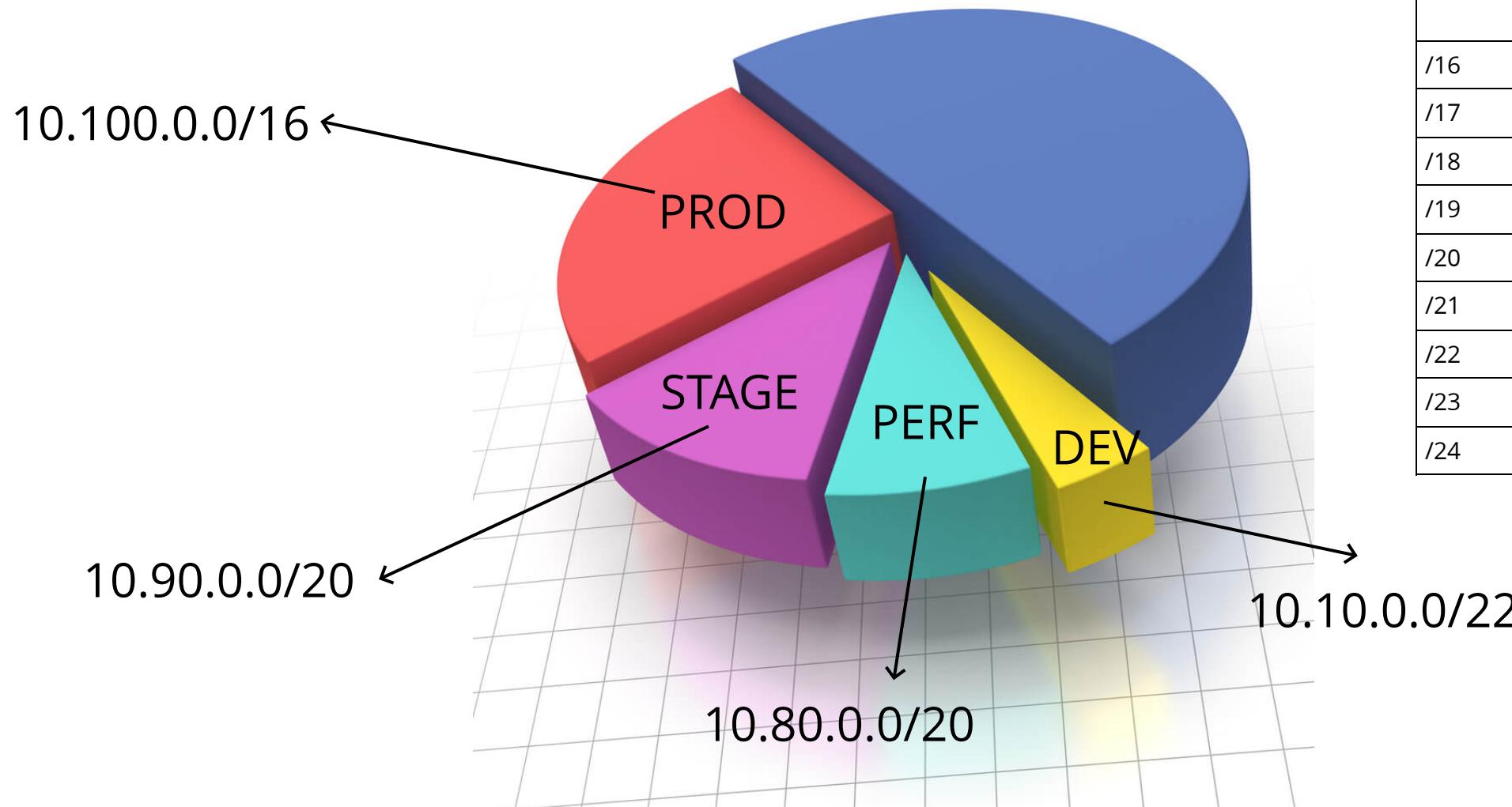
**A:** Yes, We can do that by taking all IPs which are available and consuming whatever is needed. This way we no need to forecast and start the setups immediately.

# Network IP Ranges

- Network IP range is from **0.0.0.0** to **255.255.255.255**
- These network range has been classified to give classes, Out of which first three classes are been used for end users

| <b>Class</b> | <b>Public Range ( Internet)</b> | <b>Private Range(Intranet)</b> | <b>Number of IPs</b> |
|--------------|---------------------------------|--------------------------------|----------------------|
| A            | 1.0.0.0 to 127.0.0.0            | 10.0.0.0 to 10.255.255.255     | 16581375             |
| B            | 128.0.0.0 to 191.255.0.0        | 172.16.0.0 to 172.31.255.255   | 1048576              |
| C            | 192.0.0.0 to 223.255.255.0      | 192.168.0.0 to 192.168.255.255 | 65025                |
| D            | 224.0.0.0 to 239.255.255.255    |                                |                      |
| E            | 240.0.0.0 to 255.255.255.255    |                                |                      |

# Network Subnets



Class A Subnets

| Subnet | Number of IPs | Number of Subnets |
|--------|---------------|-------------------|
| /16    | 65536         | 256               |
| /17    | 32768         | 512               |
| /18    | 16834         | 1024              |
| /19    | 8192          | 2048              |
| /20    | 4096          | 4096              |
| /21    | 2048          | 8192              |
| /22    | 1024          | 16834             |
| /23    | 512           | 32768             |
| /24    | 256           | 65536             |

# AWS VPC

- VPC is a virtual representation of network boundaries in AWS.
- It determines the size of the network
- VPC is on AWS Region Level.
- We can create multiple VPC with same network range, However we cannot make the networks communicate each other if those are same network range.
- Ideally we need to ensure the network ranges does not overlap with other because it helps in enabling inter communication when needed

# AWS SUBNETS

- SUBNET is a network that we create on the actual Availability Zone.
- While creating a subnet we need to choose to associate one AZ.
- Ideally, as a standard architecture approach, we are going to use a minimum of two AZ for High Availability purposes.
- In some cases, people use subnets for categorising business and tech needs as well, Meaning DB Subnets for databases, App Subnets for Applications, BU subnets for some business units.

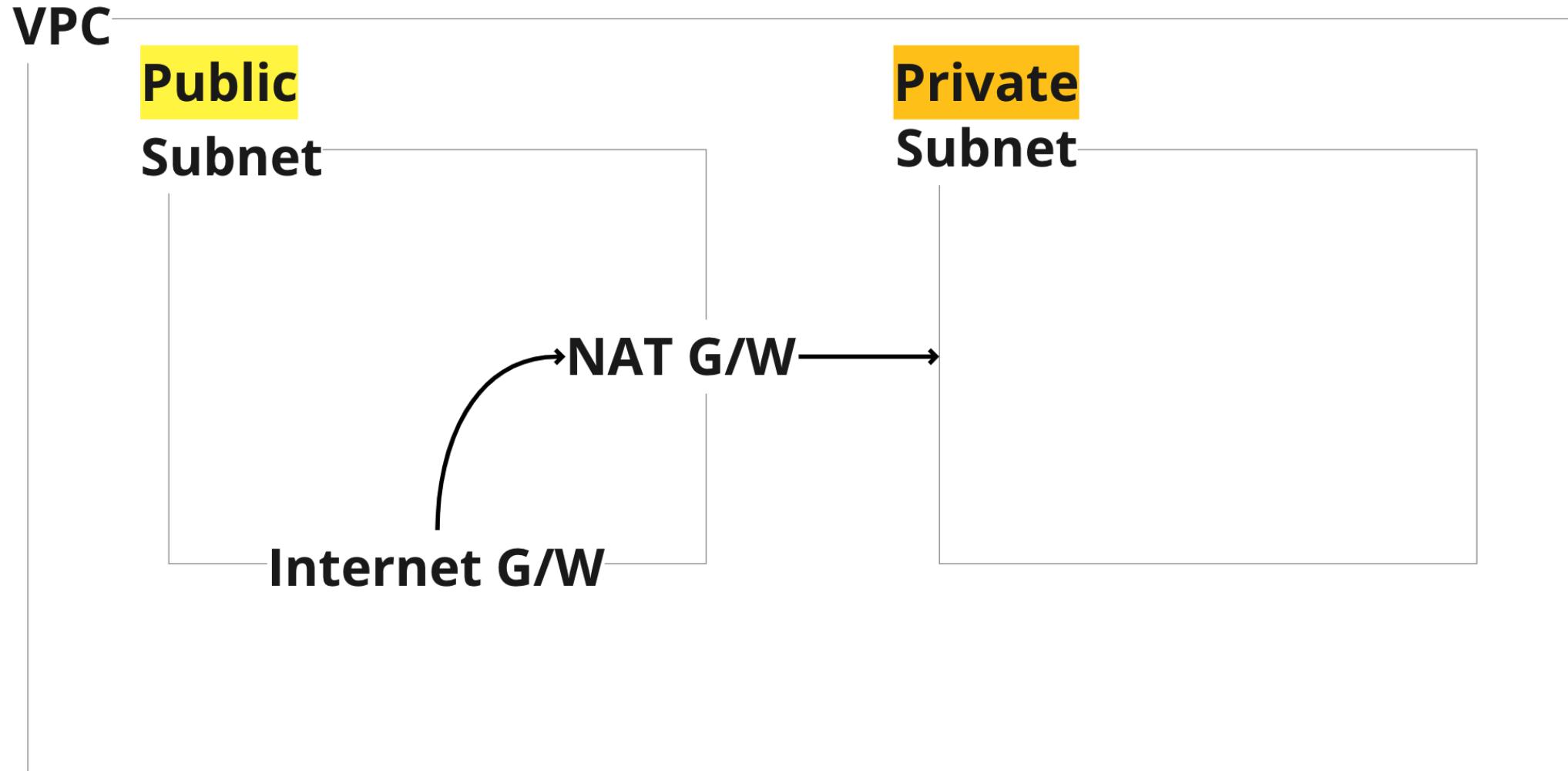
# VPC to VPC Communication

- We can communicate from one VPC to another VPC by a couple of methods.
- One, Using VPC Peering Connection. We can peer one vpc with another VPC and then we would be able to communicate each other.
- Also, In LAB we would be going with VPC Peering.
- Second, Using AWS Transit. We can communicate by adding our app vpc and default vpc to transit gw and we can communicate each other networks as well.

# VPC Peering



# Internet G/W and NAT G/W



- Subnets attached with **Internet GW** attached are called as public subnets.
- Subnets attached with **NAT GW** are called as private subnets.

# Standard VPC Structure



# AWS - RoboShop Project Architecture(Mutable)

# Containers

JAR

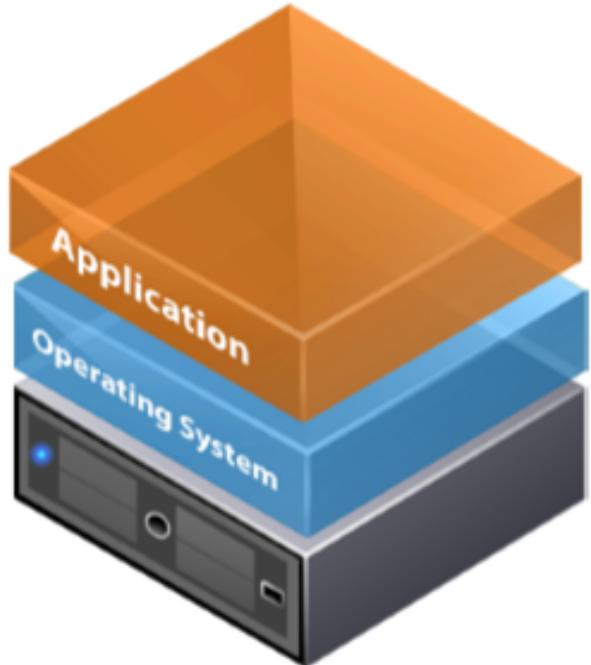
This is our interest, meaning running this application. That means how much effort reduce to run it is what is more context.

JAVA

EC2

# Compute Evolution

**Physical Machines**



**Virtual Machines**



**Container**



# Application Characteristics

|                           | <b>Physical Machines</b>                                                                                        | <b>Virtual Machines</b>                                                                         | <b>Containers</b>                                                                                            |
|---------------------------|-----------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------|
| Availability              | <b>Very less</b> , Because if the physical machine is down. The app is down, Unless you maintain more machines. | <b>Good</b> , Because VMs can move from one machine to another machine in case of H/W failures. | <b>Best</b> , Another container will be immediately created somewhere in cluster and this cause min downtime |
| Maintenance (Deployments) | CM tools -> Mutable                                                                                             | CM tools -> Immutable                                                                           | Immutable                                                                                                    |
| Cost (Operations incl)    | \$\$\$\$                                                                                                        | \$\$\$                                                                                          | \$                                                                                                           |

- Container is a Linux(Kernel) Feature.
- Means it works only on Linux OS.
- Linux Kernel offers control groups and namespaces and containers runs based on these features, Also security comes from these features

# Control Groups

Containers capable of consuming all the resources of OS, Yet we need to control or limit the resources and that part will be done by Control Groups.

# NameSpaces

Namespaces are meant to isolate the resources. For Ex, netns is a network namespace which isolates the network for containers

LXC

RKT

DOCKER

# Docker is famous, But Why?

# **Simple EchoSystem**

The whole echosystem of Docker is quite simple, Mainly the Docker Imaging part is really simple and great.

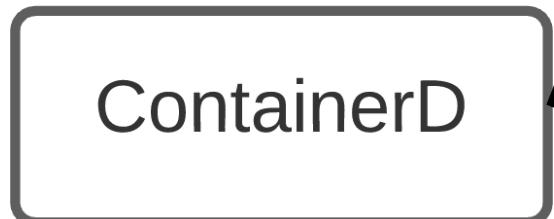
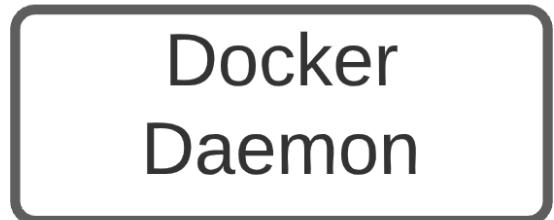
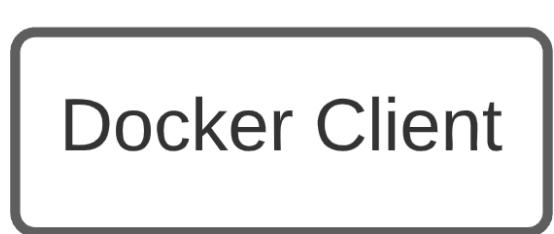
# **Best UX (User Experience)**

Docker containers are simpler the manage and extends its feature to interact over APIs.

# Docker brings Immutability

Any Container Runtime (Docker Uses ContainerD) Operations are categorized as

1. Low Level Runtime
2. High Level Runtime



1. Low Level Runtime
2. High Level Runtime

# Docker Installation

- Docker run by default with root user.
- If you give permission to normal user then he gain most of the privileges as root user. This causes serious security breaches.
- To avoid this there is a rootless installation, Where normal user can run containers even without root. This solves the security problems.

# Docker Install

```
curl -s https://get.docker.com | bash
```

# Docker Rootless Install

```
curl -s
https://get.docker.com/rootless |
bash
```

# Docker Images

To run any docker container it needs an  
Image

```
docker images
```

```
docker pull image-name
```

```
docker rmi image-name
```

# Docker Image Tags

Any docker image we pull, that image has a certain version. By default, it pulls **latest** image. In docker terminology that version is called **Tag**. **latest** is it the default tag that Docker Image uses.

Tags are used for Software releases Strategy

docker images

docker pull image-name:tag

docker rmi image-name:tag

# Docker Run

```
docker run -d nginx
```

```
docker ps
```

```
docker ps -a
```

```
docker rm
```

```
docker rm -f
```

```
docker rm -f $(docker ps -a -q)
```

```
docker exec -it <ID> bash
```

# Docker Run

- For every container we create we get a container ID & Container Name.
- We can give custom name using **--name** option.
- You can get the container low level information using ***docker inspect***

# Docker Port Expose

- Port expose can be done in two ways. Dynamic & Static Ports
- Dynamic ports ranges from 49153 to 65535
- To open dynamic port we have **-P** option
- Static port can be given with **-p** option with  
***-p <hostport>:<containerport>***

# Docker Volumes Mapping

- Containers are ephemeral and hence we lose the data if the container is deleted.
- In cases, we need to have the data persistent even if we lost or remove the container.
- A volume of host machine can be mapped to a container using -v option.  
**-v <hostpath>:<containerpath>**

# Docker Environment Variables

- We can parse some information to the containers using variables. Most of the modern applications uses variables rather than having a config file.
- We can parse environment variables either directly or from a file.
- Parse directly using **-e** option
- Parse from a file using **--env-file** option

# Container Resources

- Containers by default consume all CPU and MEM of the host machine. This is really nice feature where VMs cannot do this. At the same time, the problem arises is certain containers which are not supposed to consume the resources and start using more resources due to a bug or some performance issues then the other containers starts facing lack of resources.
- To avoid such cases we always give resource limitations and give what much is needed for that.
- **--cpus** and **--memory** are the options can be used to control the resources.

# Health Checks

- Containers have the capability to report its health & also in some criteria it has self-healing by doing an auto restart.
- If you want to restart automatically on a container exit then we can use **--restart**
- if you want to restart based on health check (Not by Docker, But by Kubernetes) then we use **--health-cmd** option

# Docker Images

# Dockerfile

# Syntax

INSTRUCTION arguments

# Docker Build

`docker build .`

`docker build -t docker.io/username/image:TAG .`

`docker build -t ghcr.io/username/image:TAG .`

# Dockerfile Reference

# Best Practices of Docker Imaging

- Size of Docker Image has to be minimal
- Security for Docker Images should have no vulnerabilities

# Who is Orchestrator?



# Kubernetes

is a Orchestrator

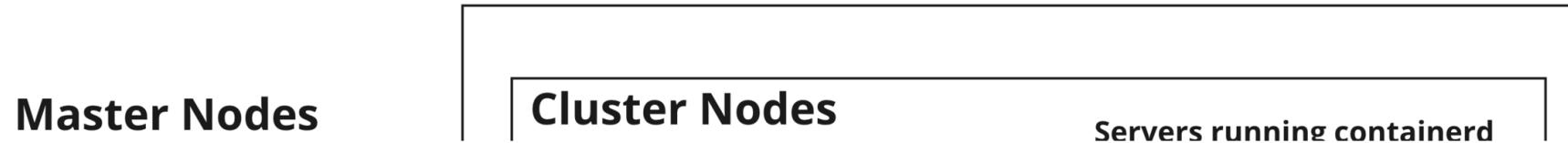
# Orchestrator

- Kubernetes
- DCOS
- Docker Swarm

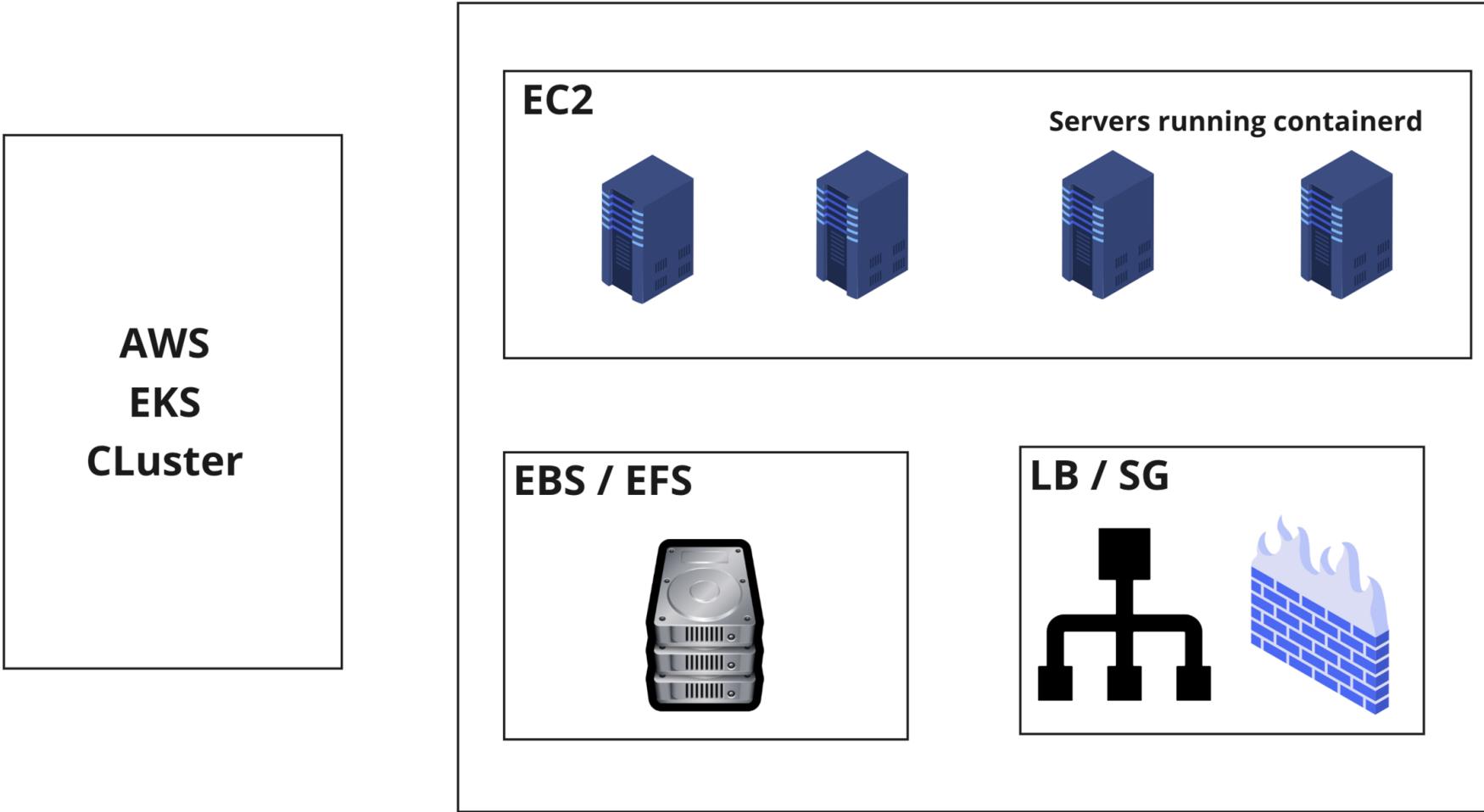
# Why Kubernetes?

1. Backed by CNCF
2. Lot of community support.
3. It is an opensource and freeware
4. Better than Docker Swarm, Solves certain problems with Network and Storage.
5. Cloud Native

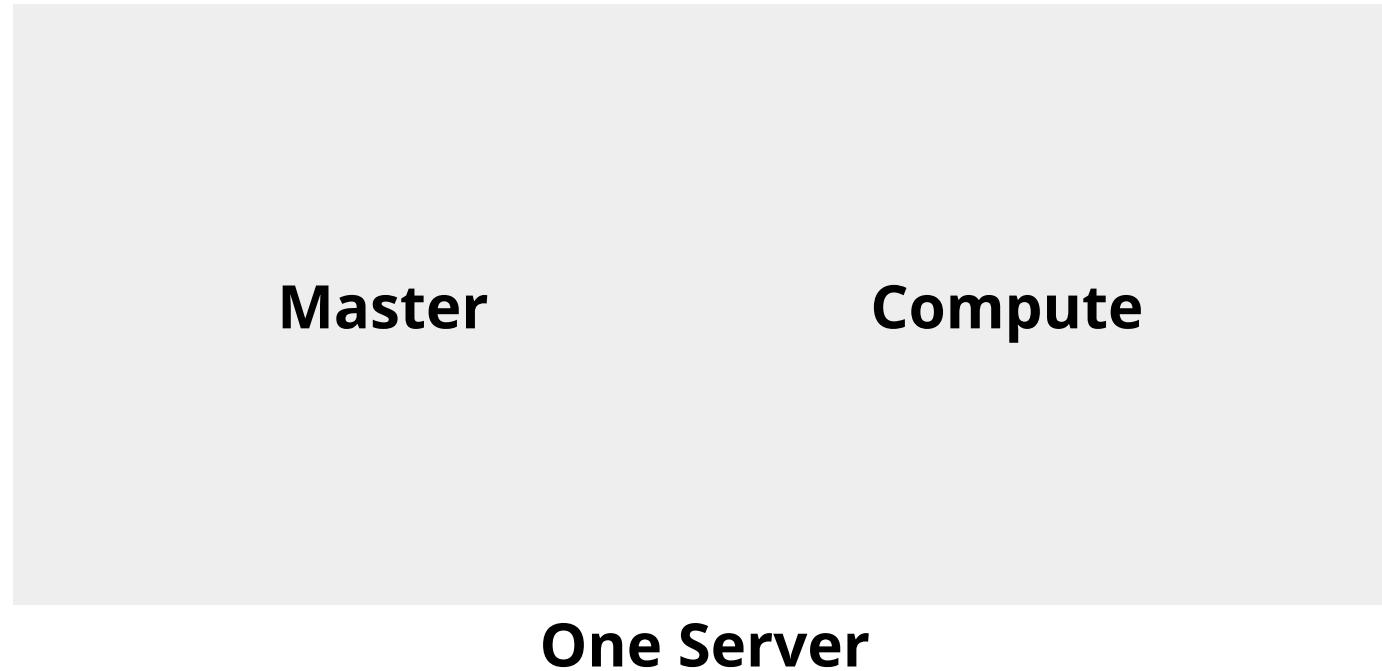
# Kubernetes Architecture



# AWS Kubernetes Architecture



# Kubernetes Minikube

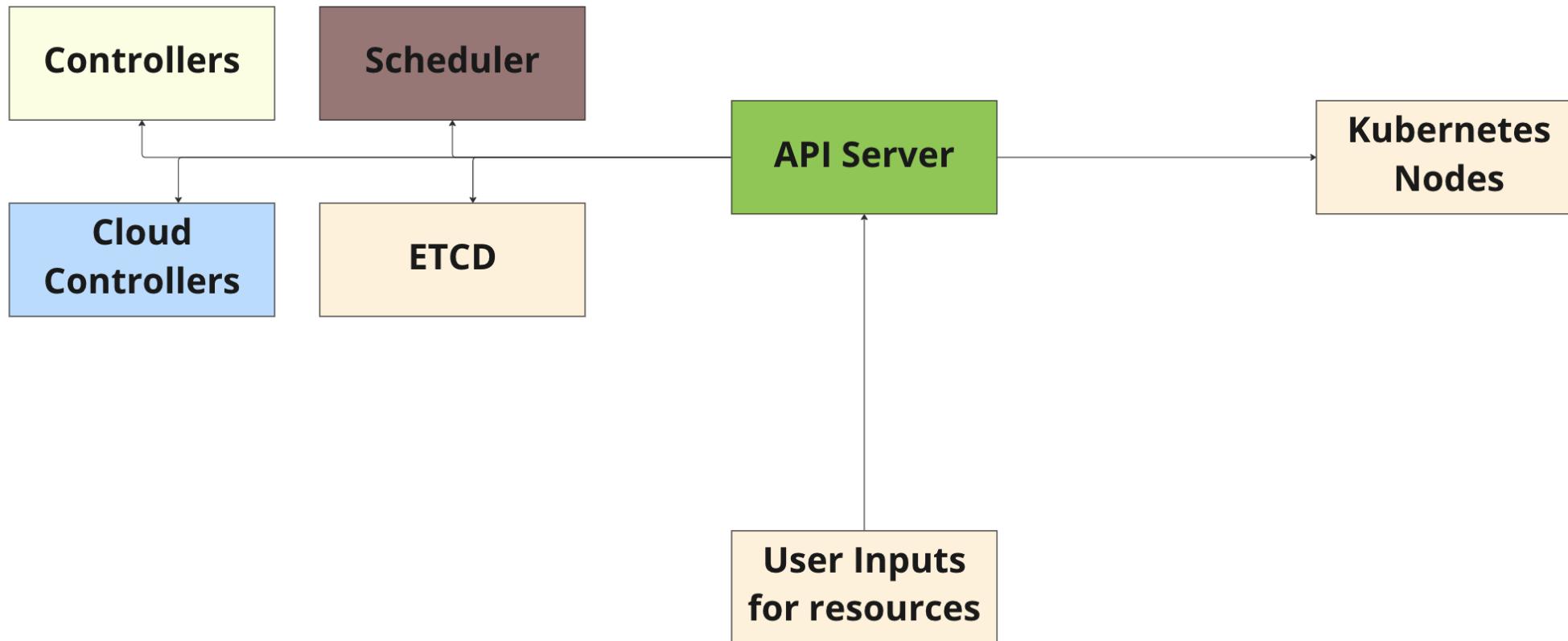


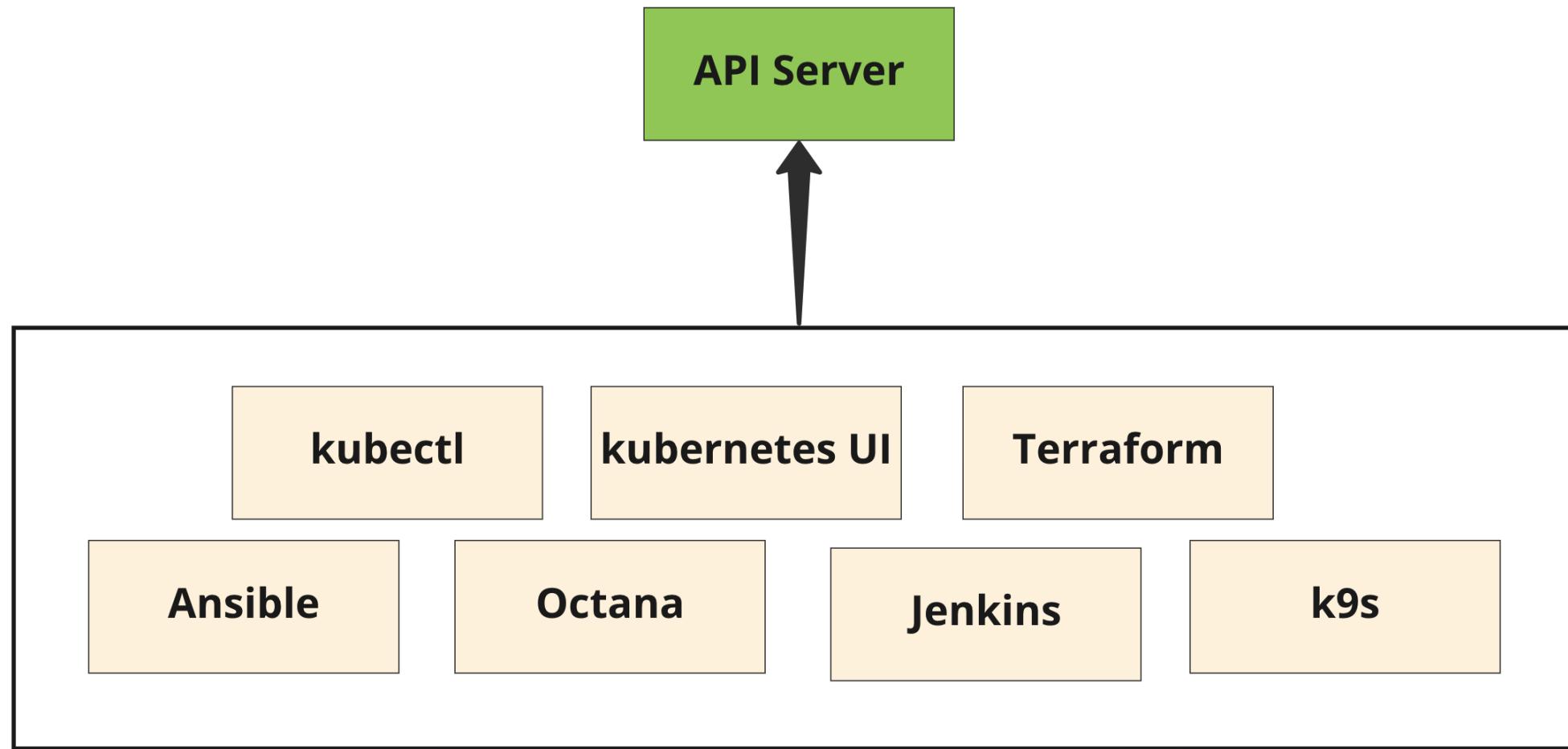
# Create Cluster

```
eksctl create cluster --name sample
--region us-east-1 --managed
```

# Kubernetes Cli Client

*kubectl*





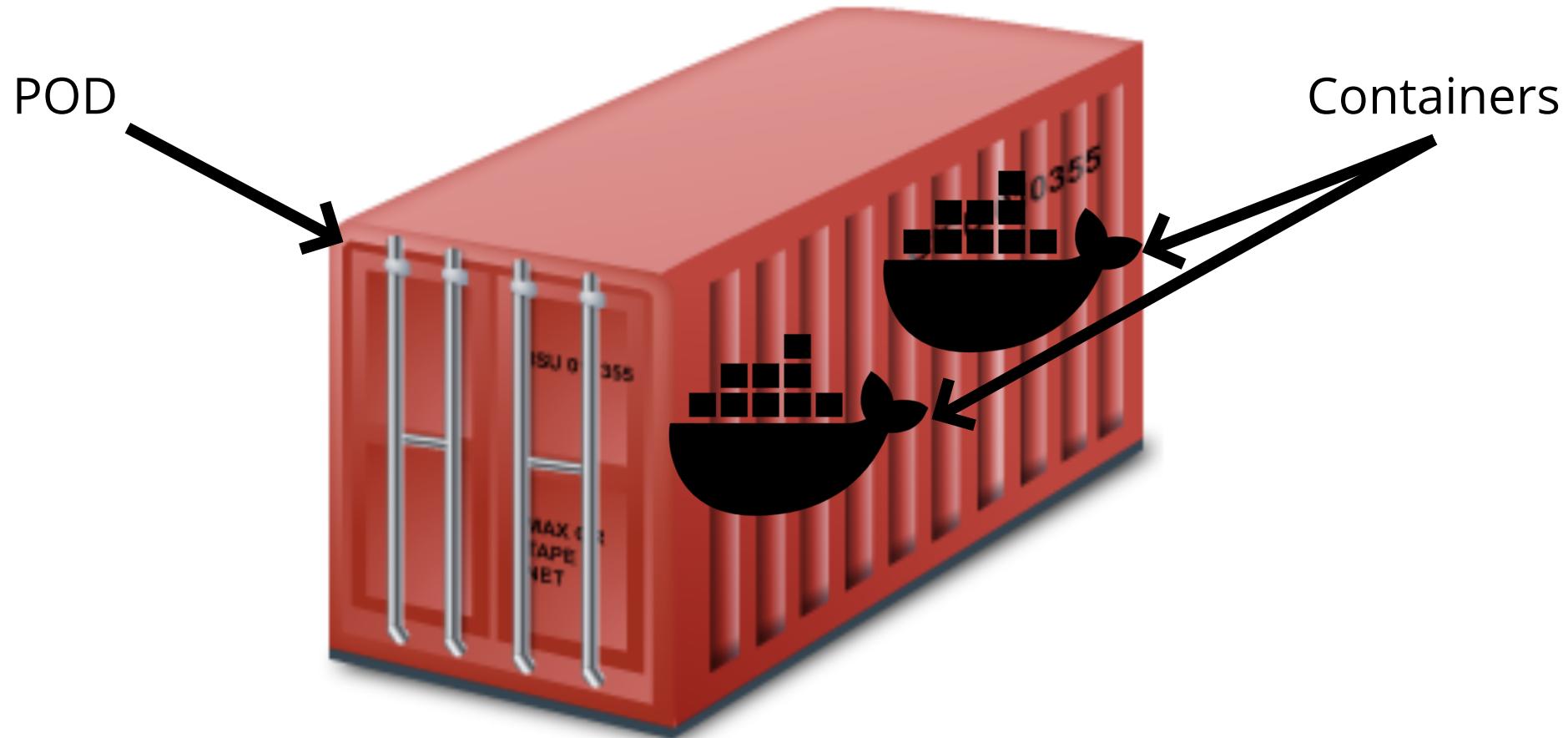
# Kubectl Commands

- **kubectl cluster-info**
- **kubectl get nodes**
- **kubectl get nodes -o wide**
- **kubectl api-versions**
- **kubectl api-resources**
- **kubectl --help**

# Kubectl Configuration

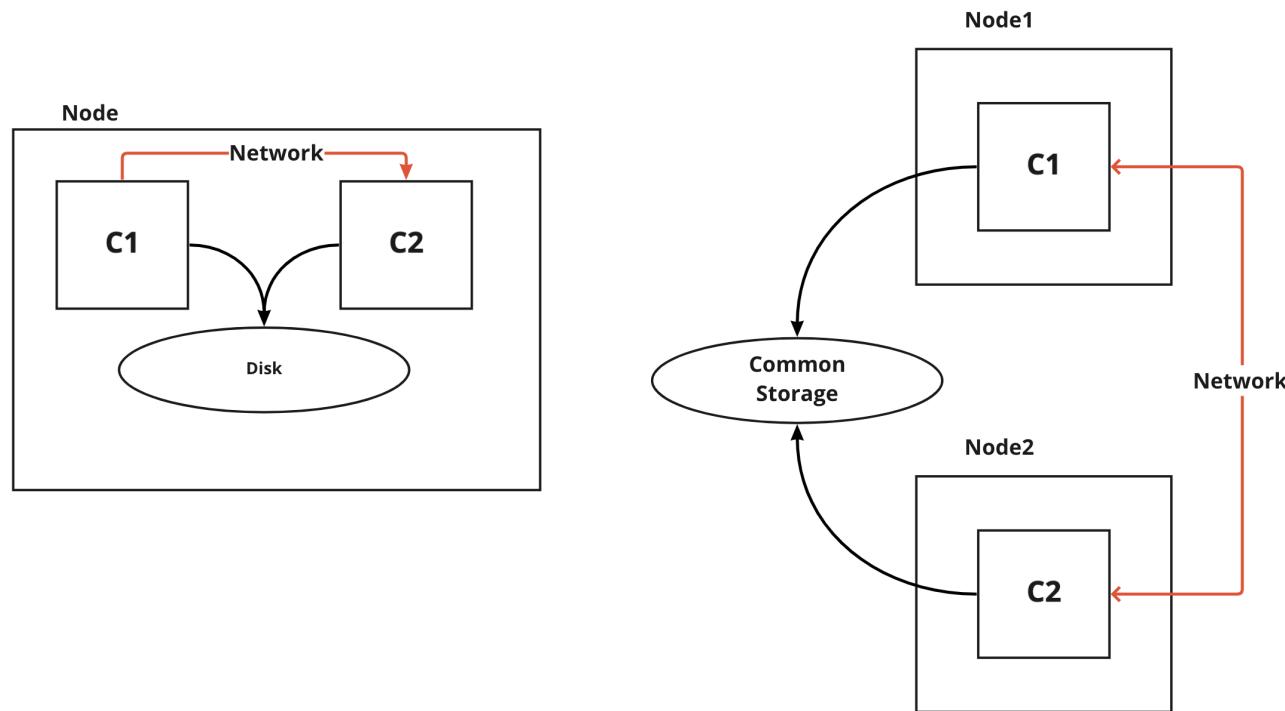
Kubectl will try to use configuration resides in home directory in *.kube/config* file

**Pod**



# What is POD

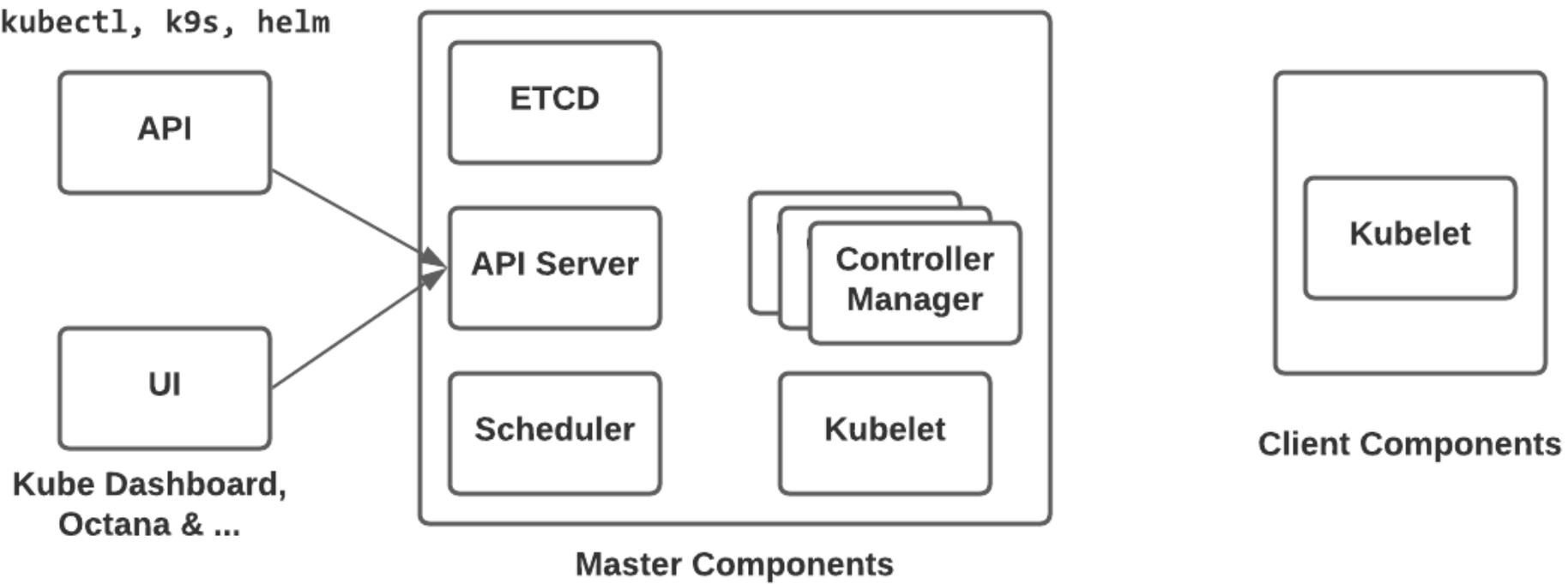
- Pod is the least unit in Kubernetes.
- Pod should have at least one container inside it. We can have one or more as well.
- Containers in Pod can share the same storage.
- Containers in Pod will use the same network stack.

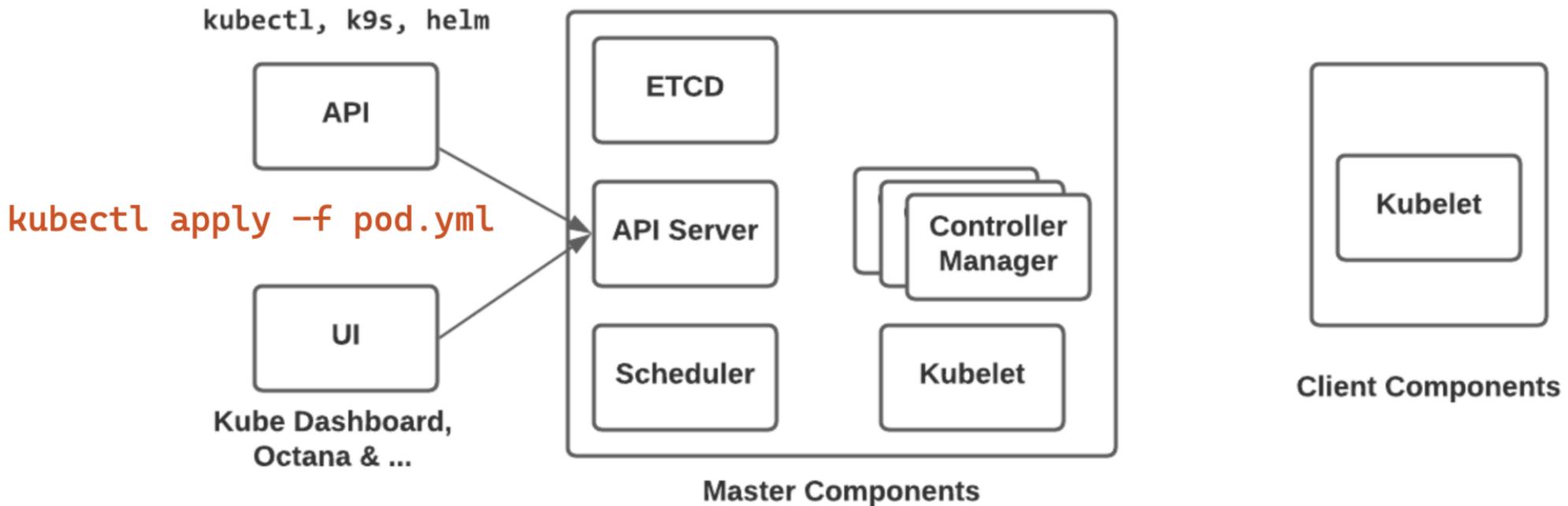


# Create Pod

```
1 apiVersion: v1
2 kind: Pod
3 metadata:
4 name: sample1
5 spec:
6 containers:
7 - name: nginx
8 image: nginx
```

# Kubernetes Components





1. Request is taken by API Server
  - a. Check auth
  - b. Checks for YAML Inputs
2. Store that a new POD creation info in ETCD DB
2. API Server talks to scheduler.
  - a. Scheduler checks all nodes and refers which node is better
  - b. API Server Updates ETCD that this is the node for new pod.
3. API Server talks to node kublet and asks to create a pod.

# Add Configs to Pod

- Using Environment Variables
- ConfigMaps can have Key Values, Config Files
- Secrets can have Key Values in encoded Formats

# Health Checks

- Kubernetes Pods can be self-healed by adding some configuration to them, That is a Health Check
- Kubernetes Supports two types of Health Checks
- **Readiness Probe & Liveness Probe.**
- In recent versions another one is also added which is **Startup Probe**

# Resources

- Kubernetes Pods can be capable of taking all the resources from the node  
(This is actually a container property)
- This leads to certain operating problems like, In case if a container is unnecessarily overutilized due to some bug then it causes problems to other containers running on the same node.
- Thus, We need control on how the resources can be allocated and that can be done on Container level using resources.
- Resources support **min** and **max** which is called as **requests** and **limits**.

# NameSpaces

- Namespaces are used to isolate pods.
- NS brings security with NetworkPolicy.
- NS can be used to allocate quotas.
- NS are generally used for organizing Pods.
- **default** is the default namespace in Kubernetes

# Set of Pods

- Usually, we don't configure directly a pod in Kubernetes.
- If we directly run then we cannot scale the Pod.
- Thus, Kubernetes offers sets and we will be using those to setup the Pods.
- **ReplicaSet, StatefulSet, DaemonSets**
- Enhancing the operation to ReplicaSet we have **Deployments** also.

# ReplicaSet

It is used for Stateless Application.  
Used to scale the pods.

# Deployment

- Deployment is a wrapper to ReplicaSet
- It helps to detect the pod spec change and can apply the changes to the pods by recreating them.
- Deployment uses Rolling Update while updating.

# **StatefulSet**

StatefulSets are for Stateful application like  
DB.

# DaemonSet

Daemonsets are pods that run on each and every node to run some process on each and every node.

Ex: Prometheus Metrics

# Statefulness with Mounts

- Containers are Ephemeral.
- Sometimes we need to deal with storing the data from container and we should not be lost even if the pod or container is terminated.
- This can be achieved by attaching storage.
- Storage can be deal in Kubernetes by **PersistentVolume**, **PersistentVolumeClaim** & **StorageClass**

PRIVATE-  
SUBNET

RABBITMQ

EKS

FRONTEND

CART

CATALOGUE

USER

SHIPPING

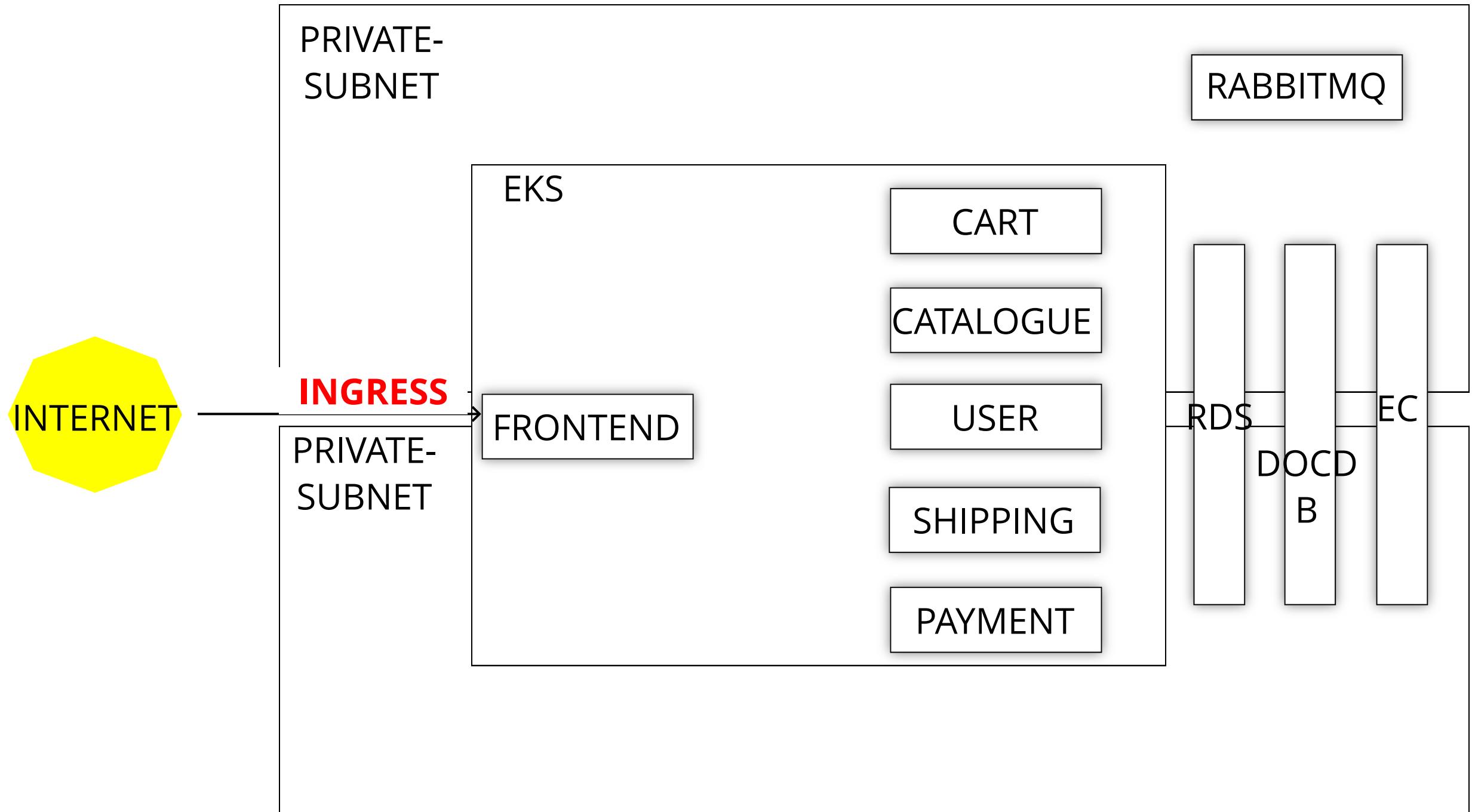
PAYMENT

RDS

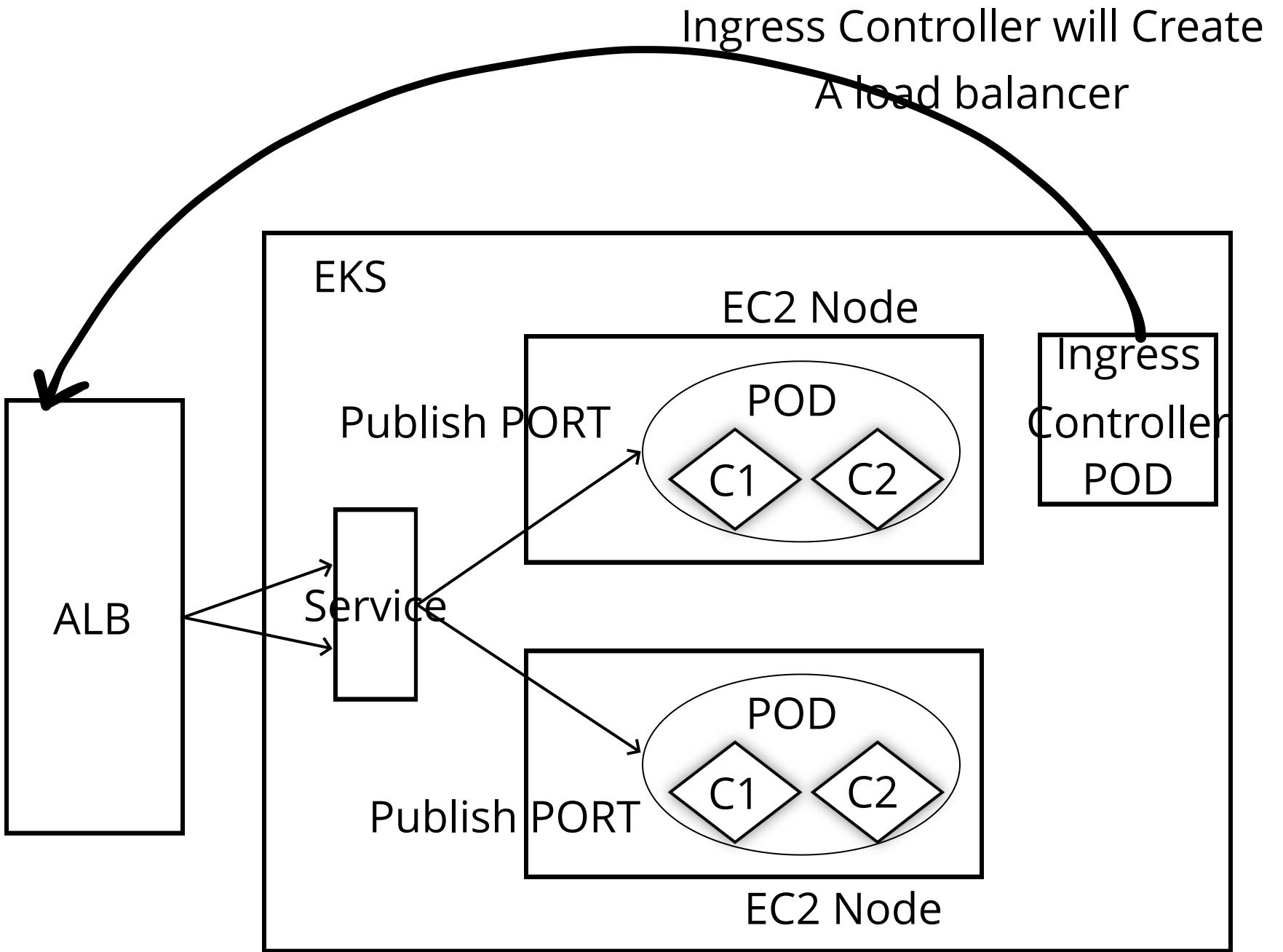
DOCDB  
B

EC

PRIVATE-  
SUBNET



What Ingress does?

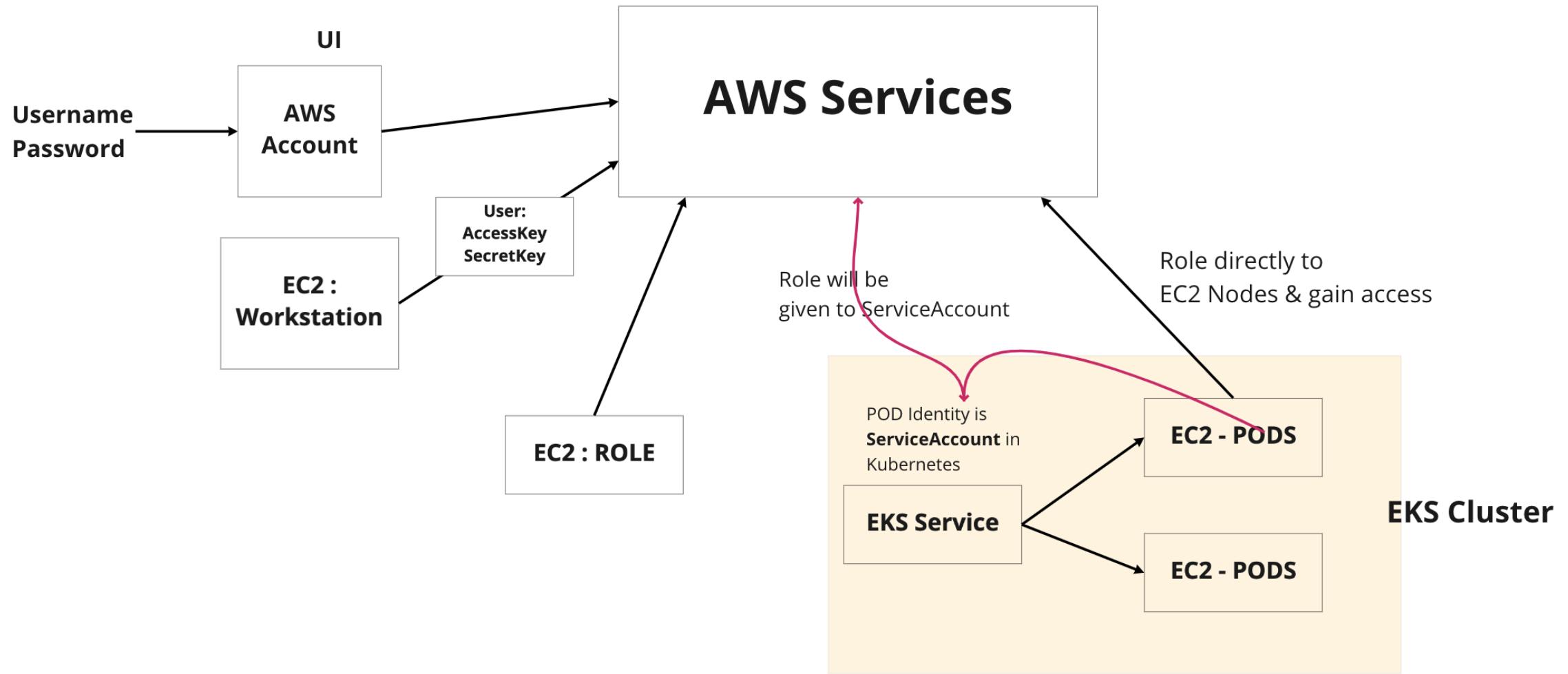


# Networking

- In Kubernetes, we can expose the service running inside a Pod in multiple ways.
- Depends on the need of exposing whether it can be internal or to external, Kubernetes has different network resources.
- **ClusterIP, NodePort, HostPort, LoadBalancer**
- Kubernetes Networking works on the DNS layer. This solves a lot of network complexity.

|           | <b>General</b>                                                                                     | <b>RoboShop</b>                                                                           |
|-----------|----------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------|
| ClusterIP | This is internal, It is used for internal pod communications. This is a default one in kubernetes. | Yes, Frontend needs to talk to catalogue that will happen internally and we use ClusterIP |
|           |                                                                                                    |                                                                                           |

# Identity



# External Secrets