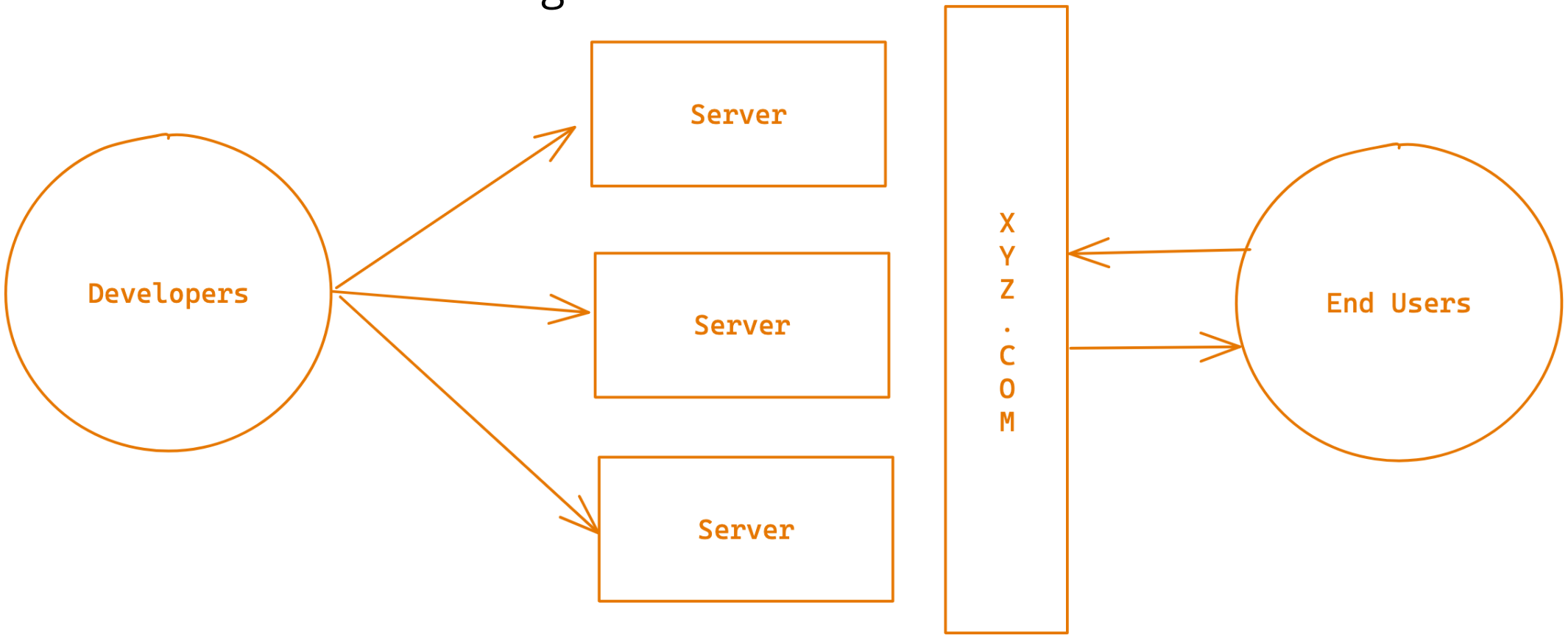
Roles of a DevOps Engineer

- 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 cane be modified & Can be re-distributed.

Who redistributes

- **CentOS**
- Oracle
- Amazon

Which CentOS

- **7 (Actively Highly Used one)**
- 8

Where to run Linux?

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

Linux Basics

File Commands

```
ls

touch

rm

cp

mv
```

Directory Commands

```
pwd

cd

mkdir

rmdir

rm -r

cp -r

mv
```

File Content Commands

```
cat

head

tail

grep

awk

vim
```

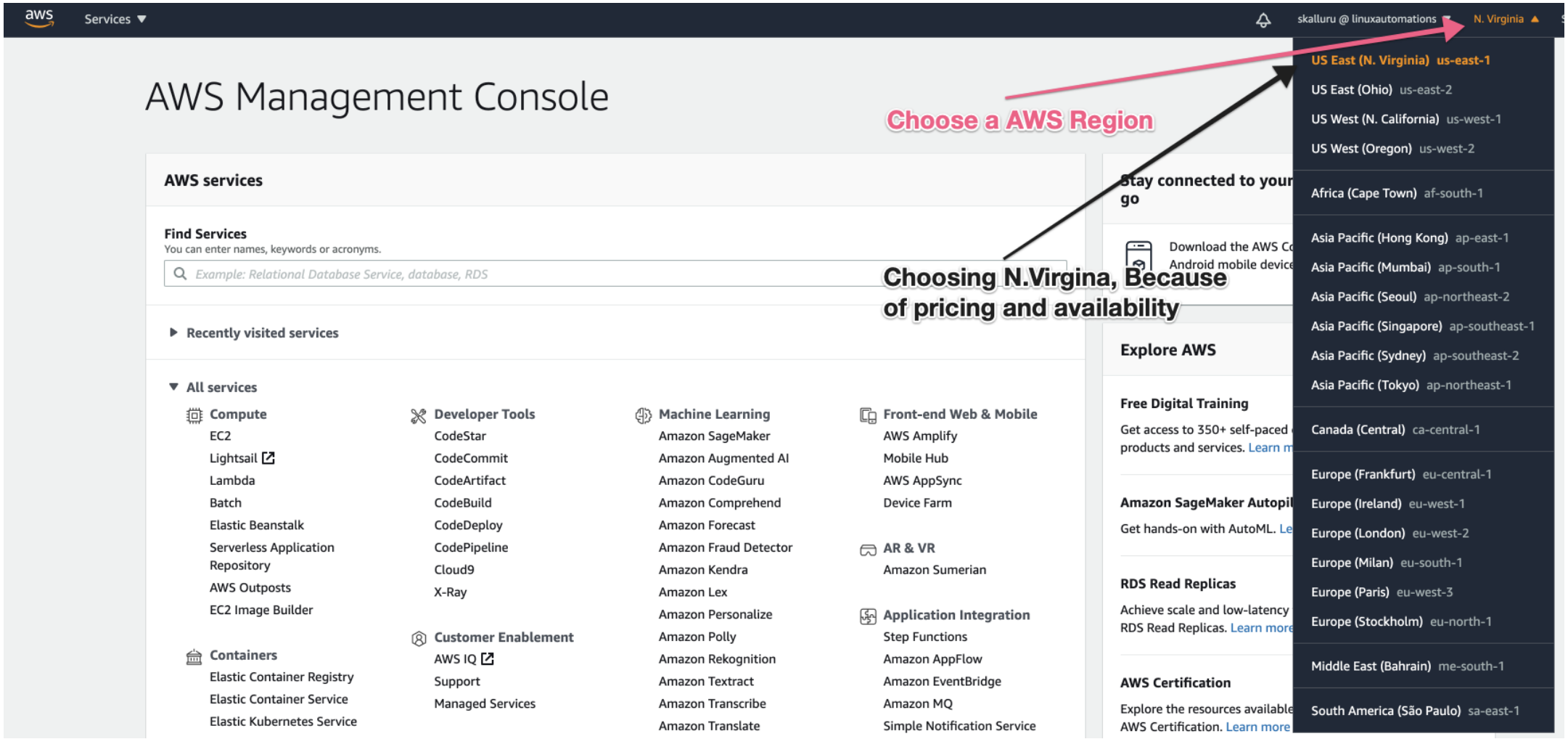
Utility Commands

```
find

curl

tar

unzip
```

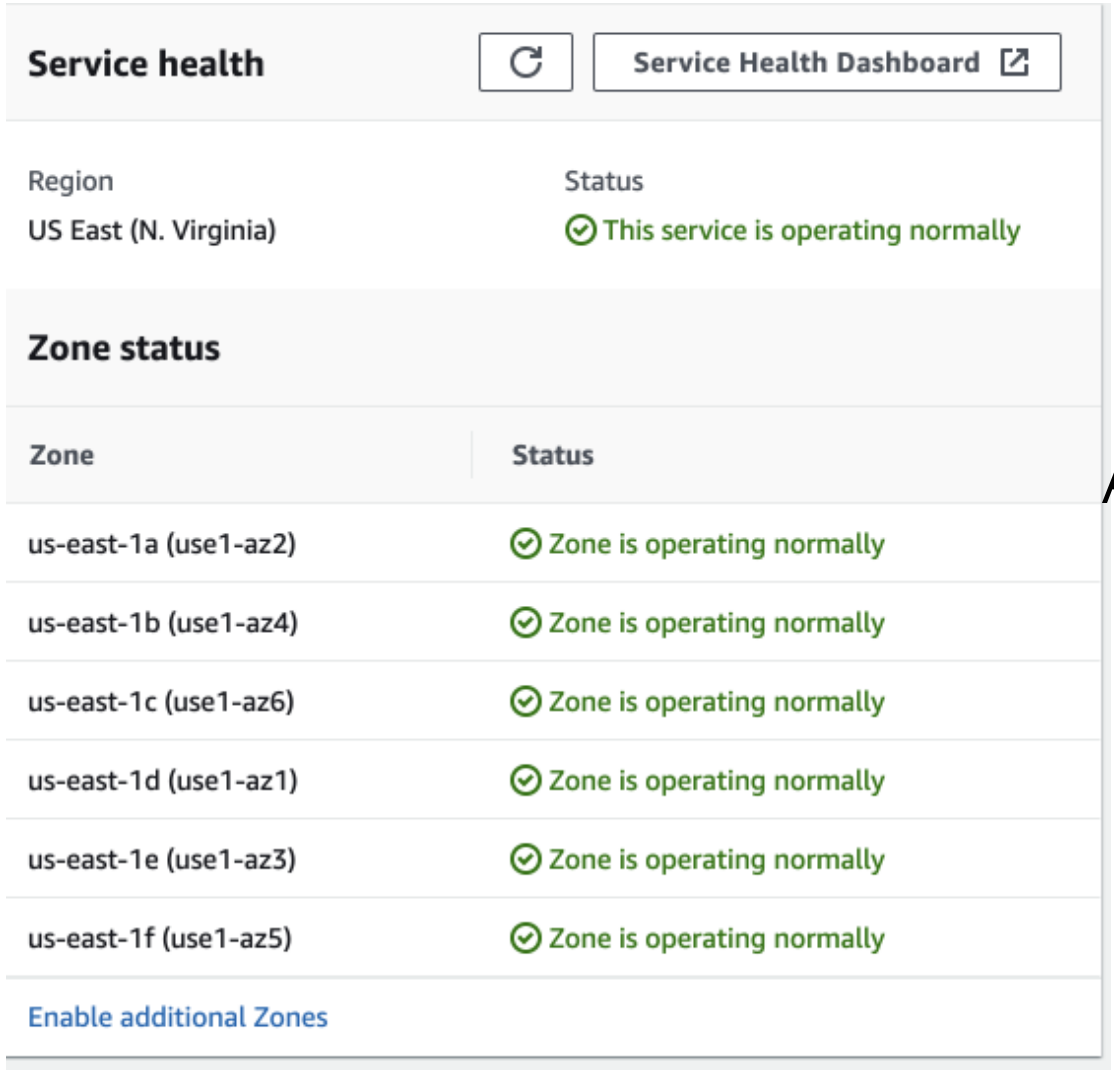
Pipes (|)

AWS Account

Choose Region

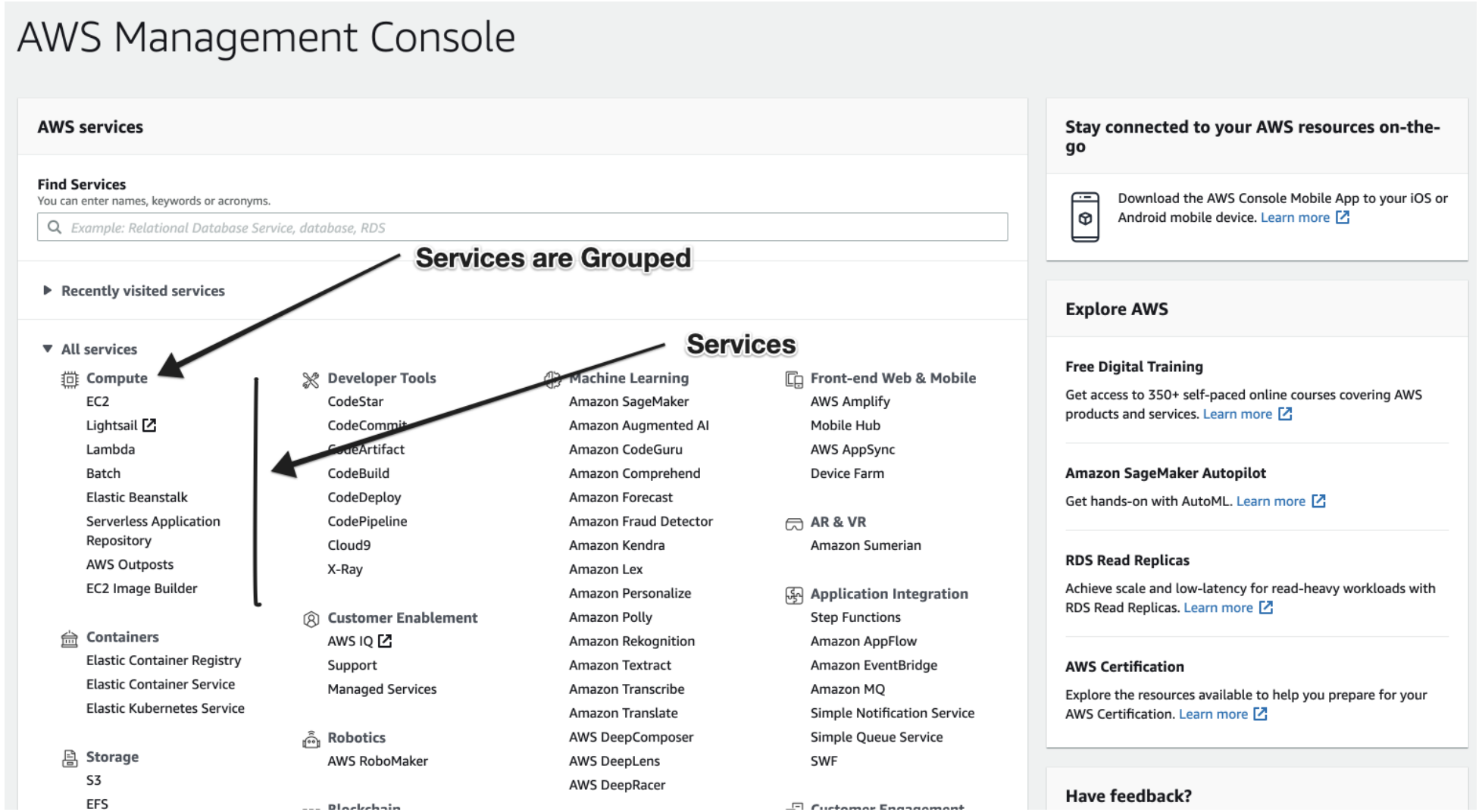


Availability Zones



AZ is nothing but one data center

Services


AWS Management Console
AWS services
Find Services
You can enter names, keywords or acronyms.
Example: Relational Database Service, database, RDS
Services are Grouped
Services
Recently visited services
All 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
AWS DeepComposer
AWS DeepLens
AWS DeepRacer
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
Stay connected to your AWS resources on-the-go
Download the AWS Console Mobile App to your iOS or Android mobile device. Learn more
Explore AWS
Free Digital Training
Get access to 350+ self-paced online courses covering AWS products and services. Learn more
Amazon SageMaker Autopilot
Get hands-on with AutoML. Learn more
RDS Read Replicas
Achieve scale and low-latency for read-heavy workloads with RDS Read Replicas. Learn more
AWS Certification
Explore the resources available to help you prepare for your AWS Certification. Learn more
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
```

```
yum install
yum remove
yum update
```

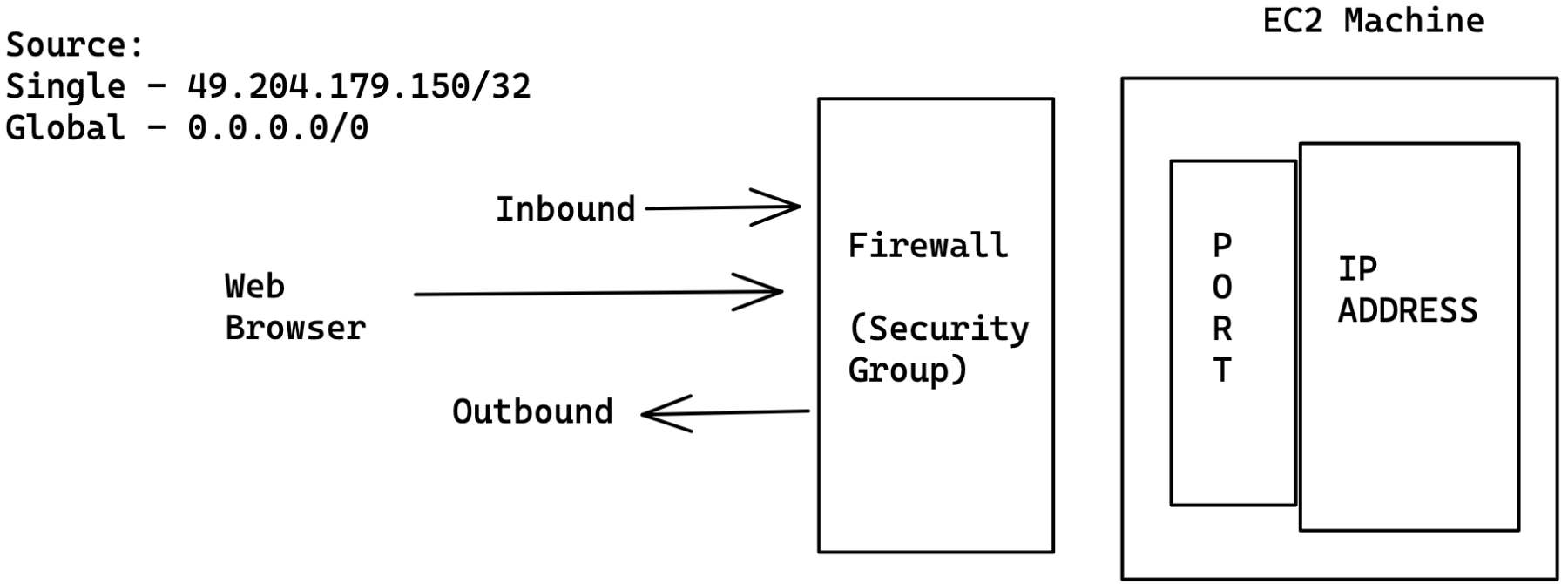
```
chmod

chgrp

chown
```

```
groupadd

useradd

usermod

visudo

sudo

su
```

```
systemctl enable
systemctl start
systemctl stop
systemctl restart
```

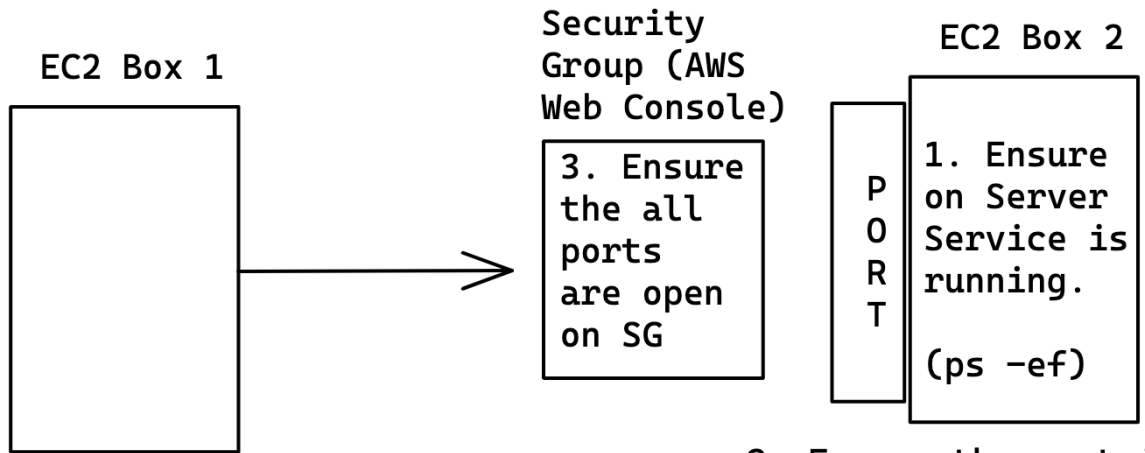
```
ip a
netstat -lntp
telnet
```

Firewall



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



2. Ensure the port is opened by the service. (netstat -lntp)

4. In cases if that service is accessed by another server, then you can check the reachability using

(telnet IP-ADDRESS PORT)

5. Restart the services on both the servers
6. Repeat step 4.
7. Finally you can post on slack

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

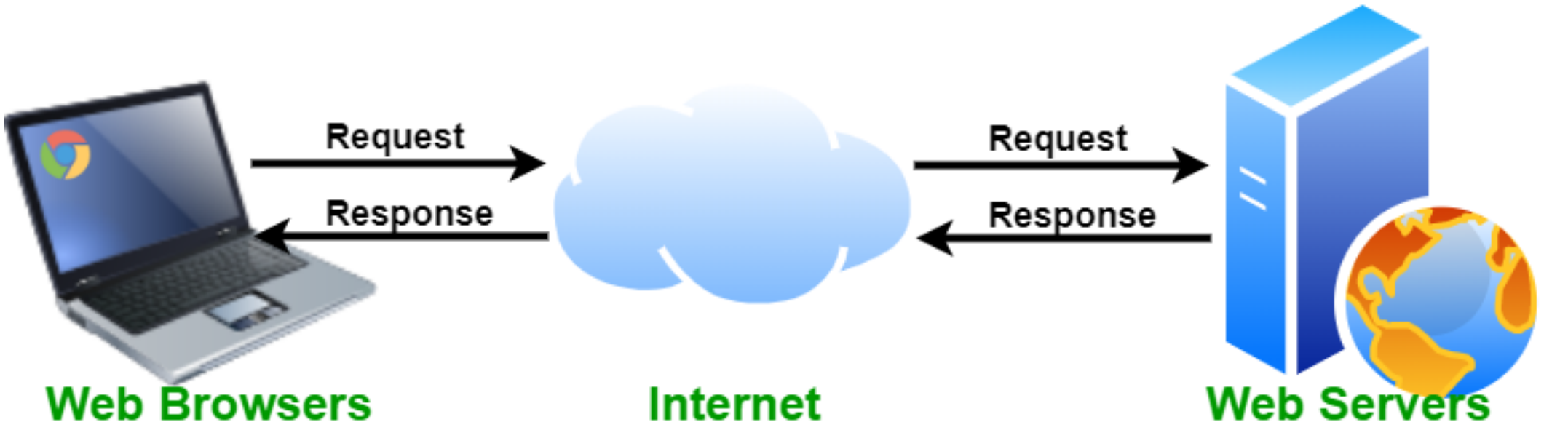
Amazon

Twitter

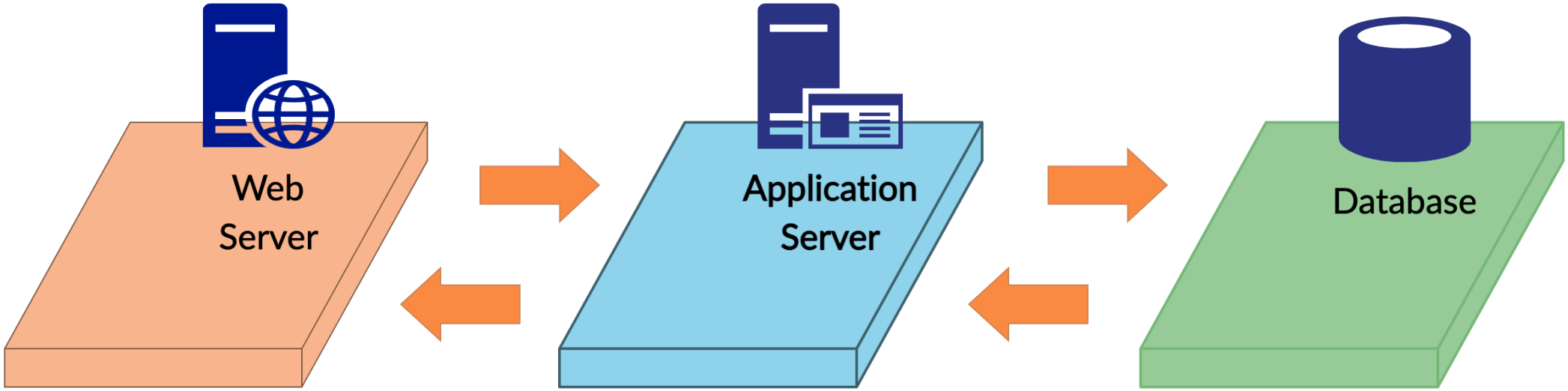
LinkedIN

How you get web content in Modern Applications?

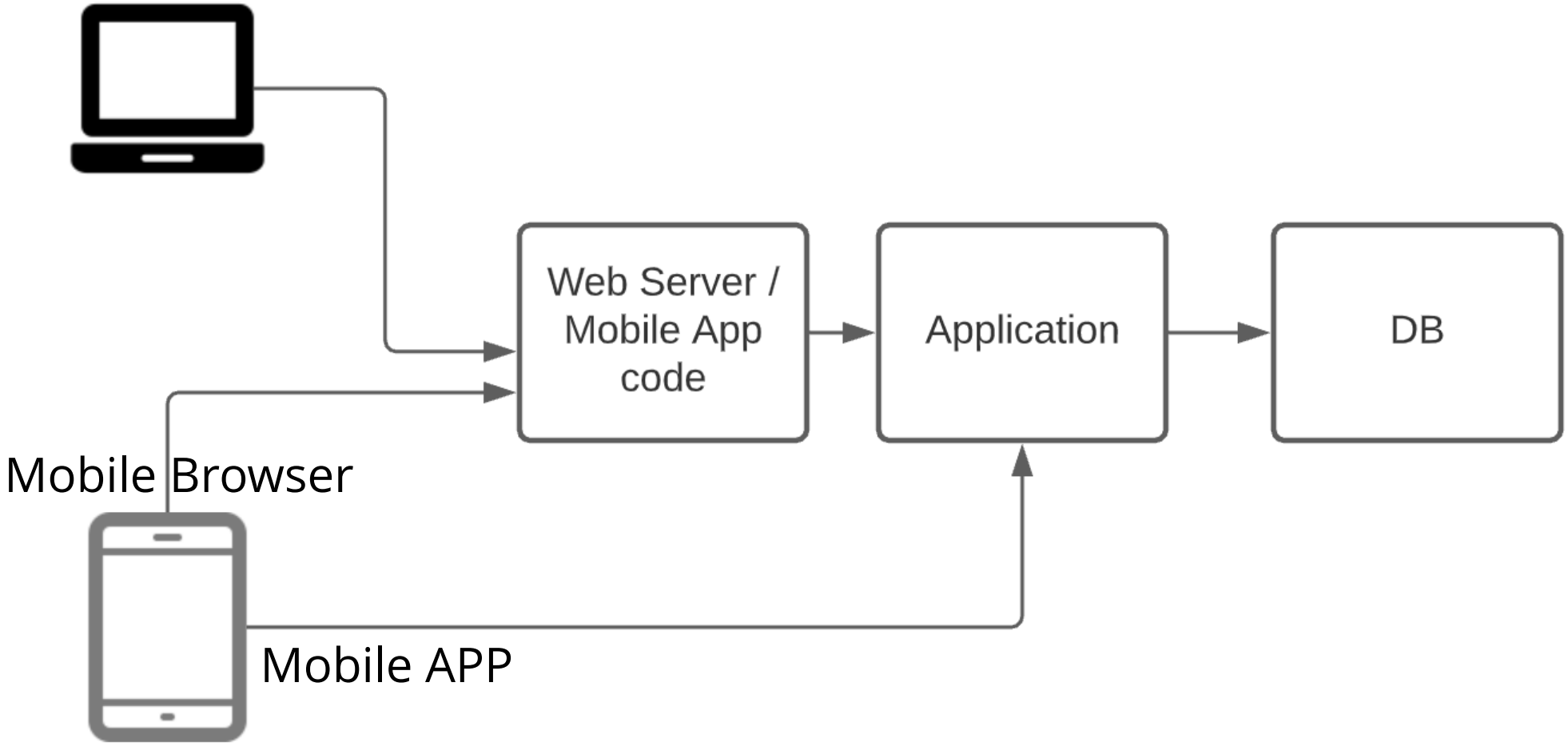
WebContent


Request
Response
Request
Response
Web Browsers
Internet
Web Servers

Web Application



3-Tier Architecture


Web Server / Mobile App code
Application
DB
Mobile Browser
Mobile APP

Web Server

- Setup
- Host a Website
- Reverse Proxy
- Redirect rules

Which Nginx Server?

Who says which version? Developers.

HTTP Status Codes

1XX
INFORMATIONAL

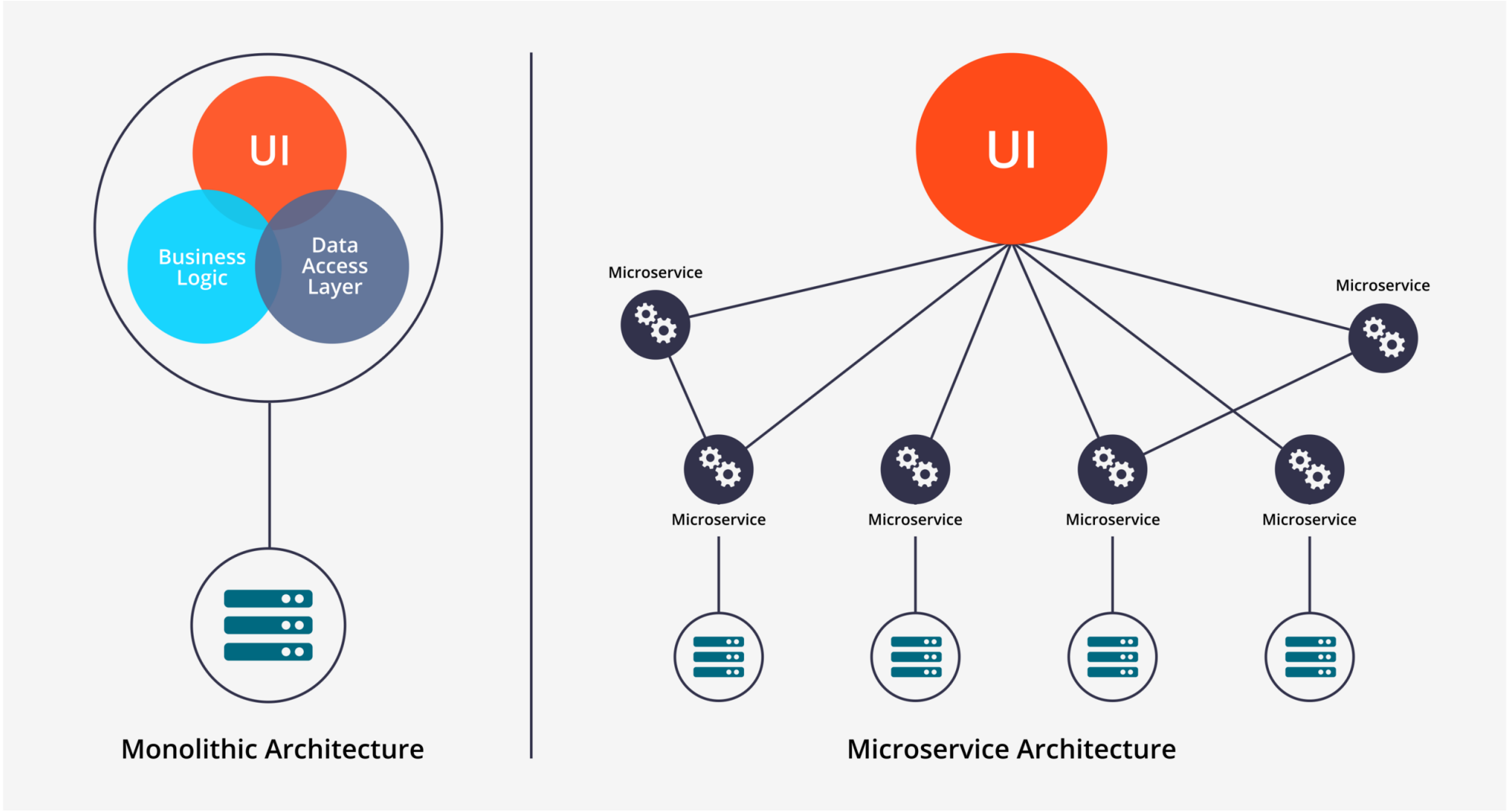
2XX
SUCCESS

3XX
REDIRECTION

4XX
CLIENT ERROR

5XX
SERVER ERROR

Monolith vs Microservices

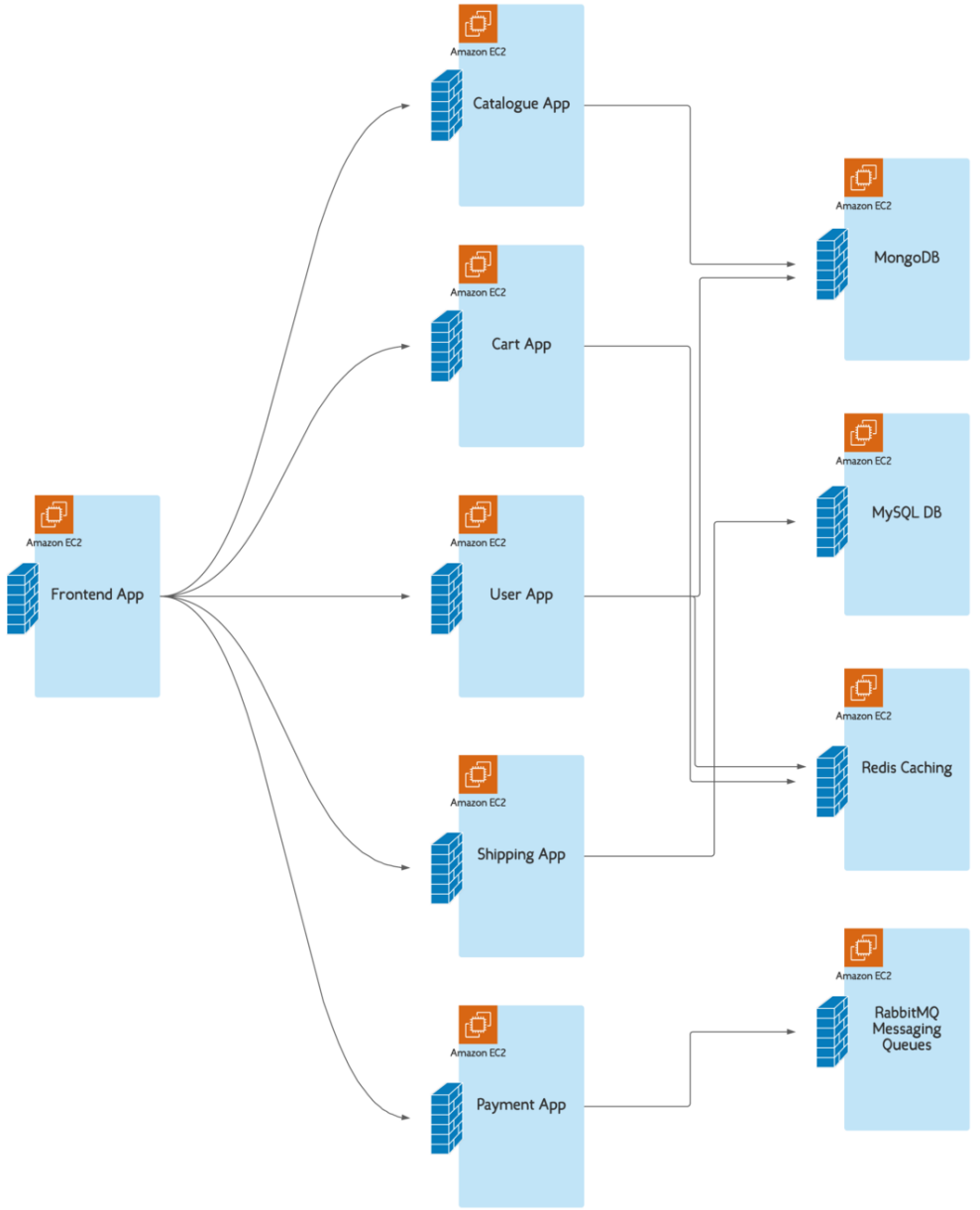


Advantages of MicroServices

- 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)

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.

AWS Spot Instances

Instance Types

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

EC2 Launch Templates

Setup RoboShop Application

Problems we can Forecast

- Security
- **Manual Effort / Repetitive Effort**
- DR & Backups
- Tracing the issue
- Scalability
- High Availability
- Log aggregation & Monitoring

Automation (Scripting)

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

Code

Code Standards

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

- Code should never hardcode the username and passwords.

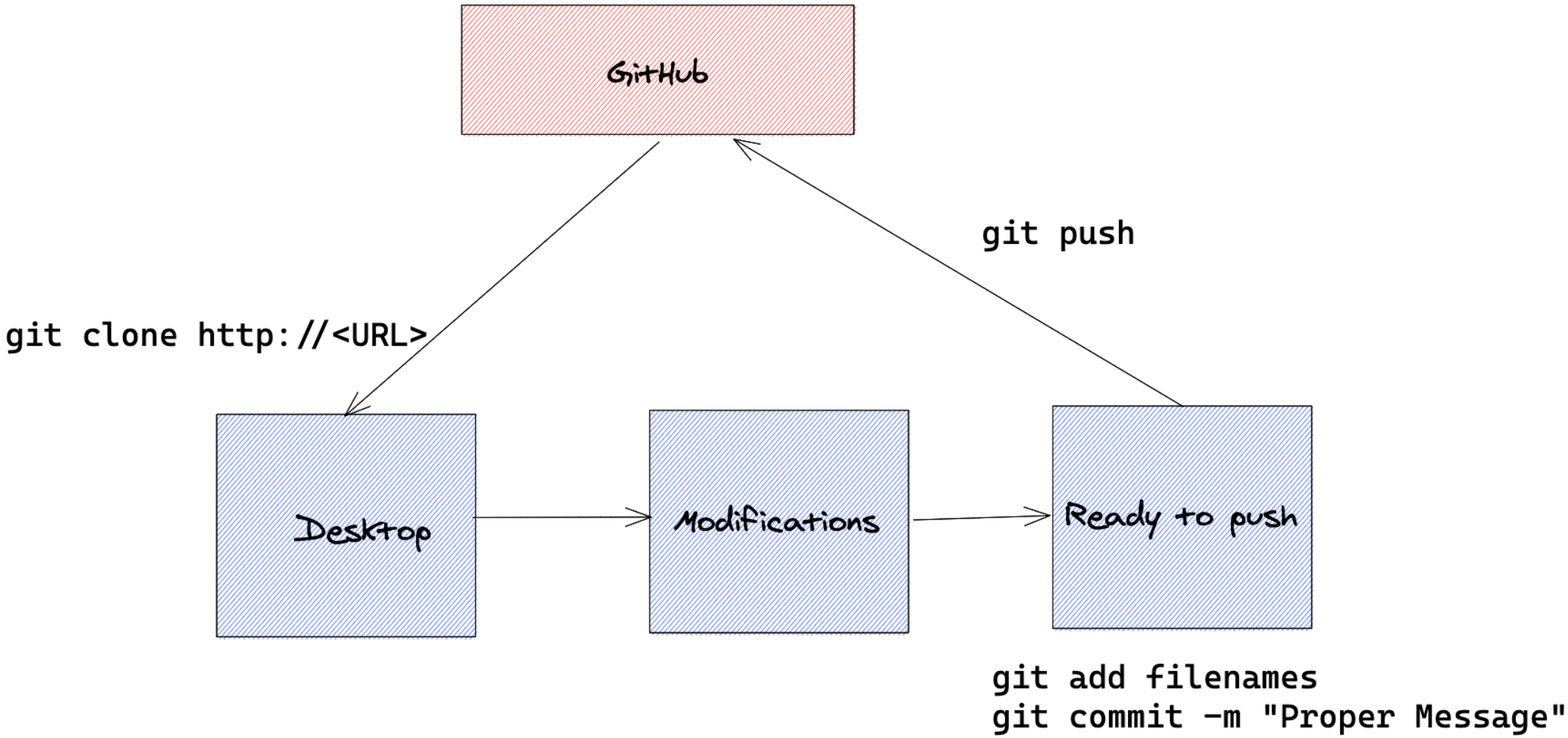
VCS (Version Control System)

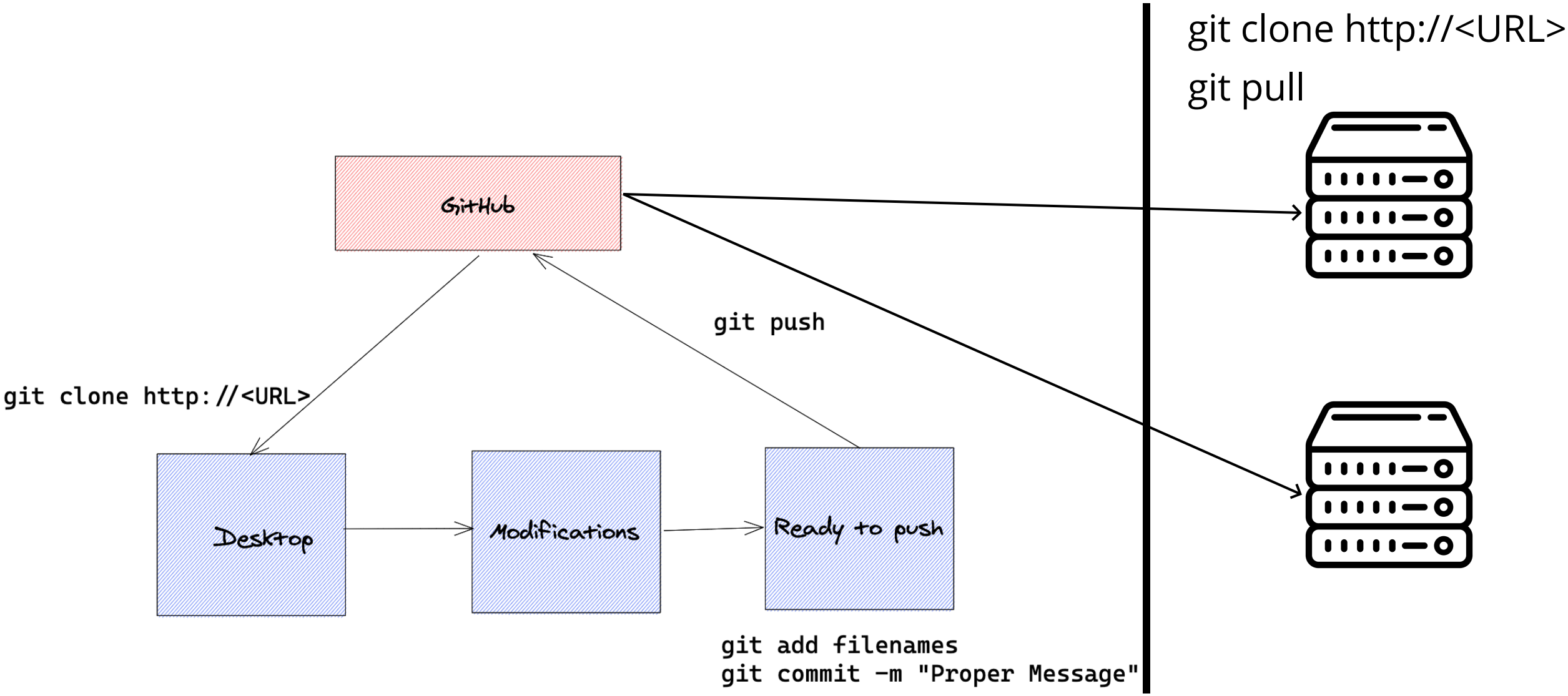
- Maintaining your code in version manner.
- SVN & GIT are famous tools.
- **GIT** is highly used on modern development.

GIT

Git in Central?

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


GitHub
git push
git clone http://<URL>
Desktop
Modifications
Ready to push
git add filenames
git commit -m "Proper Message"


git clone http://<URL>
git pull
GitHub
git push
git clone http://<URL>
Desktop
Modifications
Ready to push
git add filenames
git commit -m "Proper Message"

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

Code Best Practices

- Code WET vs **DRY** - Code should always be DRY
- Try to make the best design of code structure for your requirement.
- Repetetive run of shell script should not fail.

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
```

Comments

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

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

- 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: COURSE_NAME**
- People from Java/DEV background prefer CamelCases Variables
  Ex: courseName / CourseName

Variables

- Variables of Bash Shell will hold three properties

1. ReadWrite

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

2. Scalar

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

3. Local

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

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

$
```

ReadOnly

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

Arrays

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

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

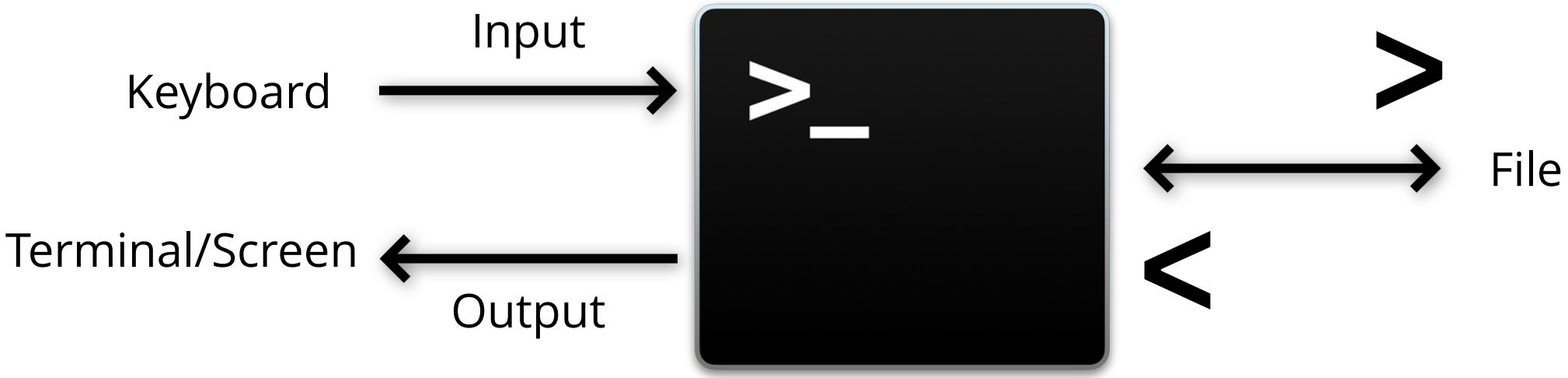
Special Variable	**Purpose (To get what)**	**Values from Example**
$0	Script Name	script1.sh
$1	First Argument	10
$2	Second Argument	abc
$*	All Arguments	10 abc 124
$@	All Arguments	10 abc 124
$#	Number of Arguments	3

Quotes

Single Quotes	**Does not consider any character as a special character**
Double Quotes	**Very few characters like $ will be considered as special and remaining of them are normal characters**

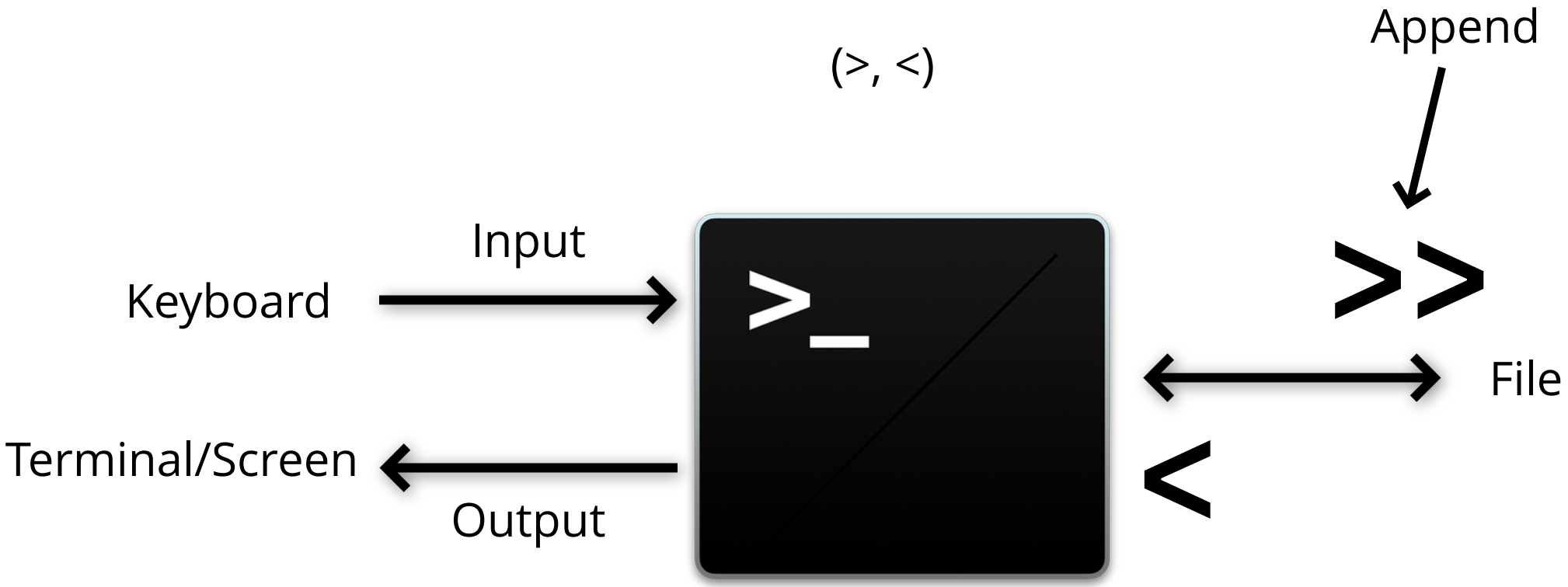
Redirectors

(>, <)



Redirectors

(>, <)



Redirectors

(>, <)

STDOUT (>)

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

STDIN (<)

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

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

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

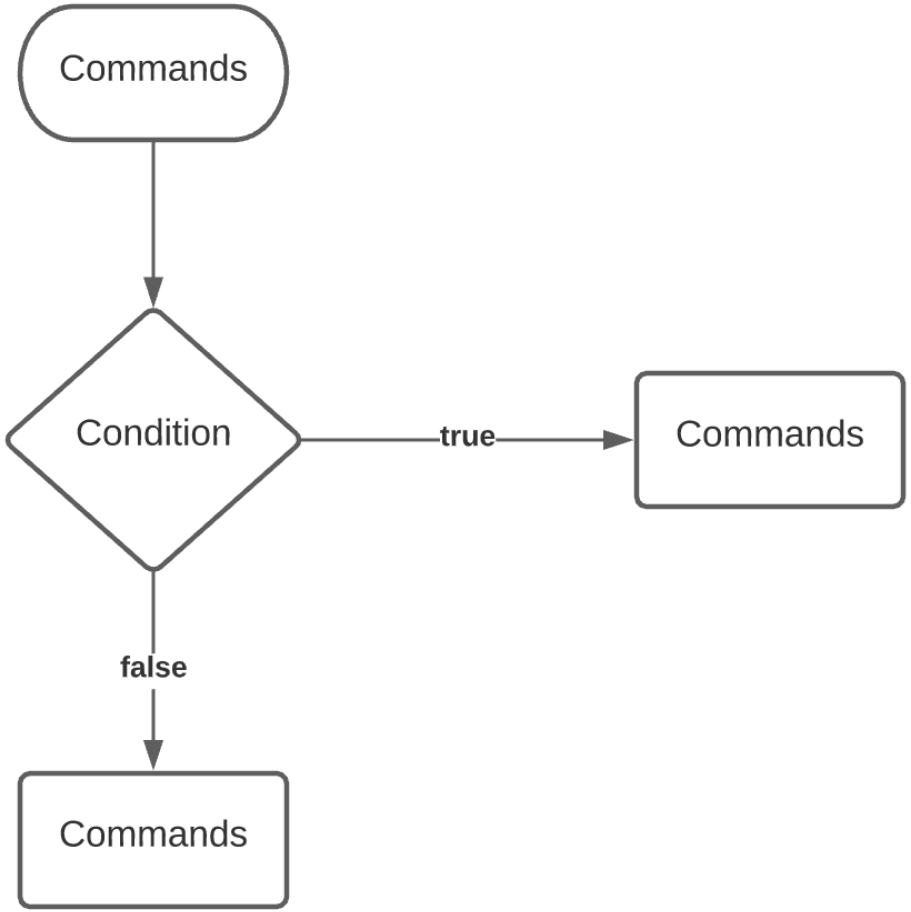
- In times we need to come out of function to the main program, but we cannot use **exit** command because exit will completely exit the script.
- **return** command will be used to come out of function.
- Function is also a type of command which necessarily needed the exit status. So **return** command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

Project Code Structure

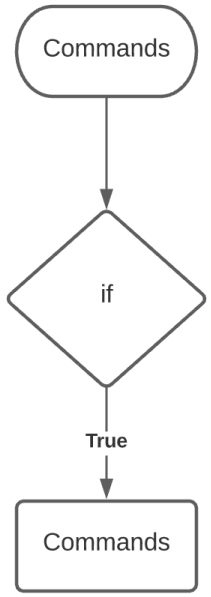
1. Individual scripts for all components of the project.
2. **Individual scripts for all components wrapped with Makefile**
3. Individual scripts for all components wrapped with own shell script.
4. All components are written in one script.

Conditions

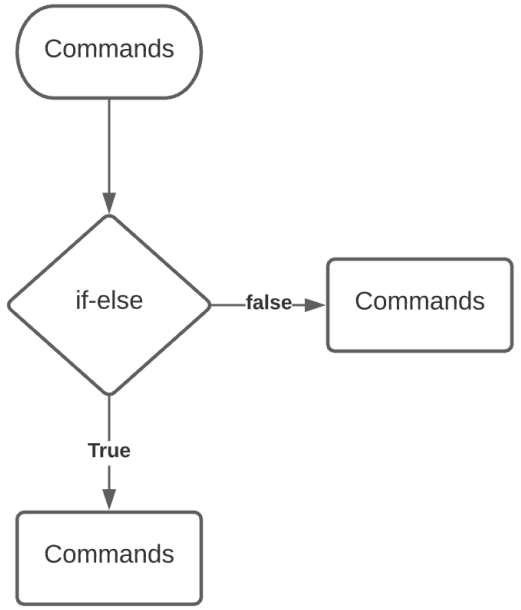
- case
- if



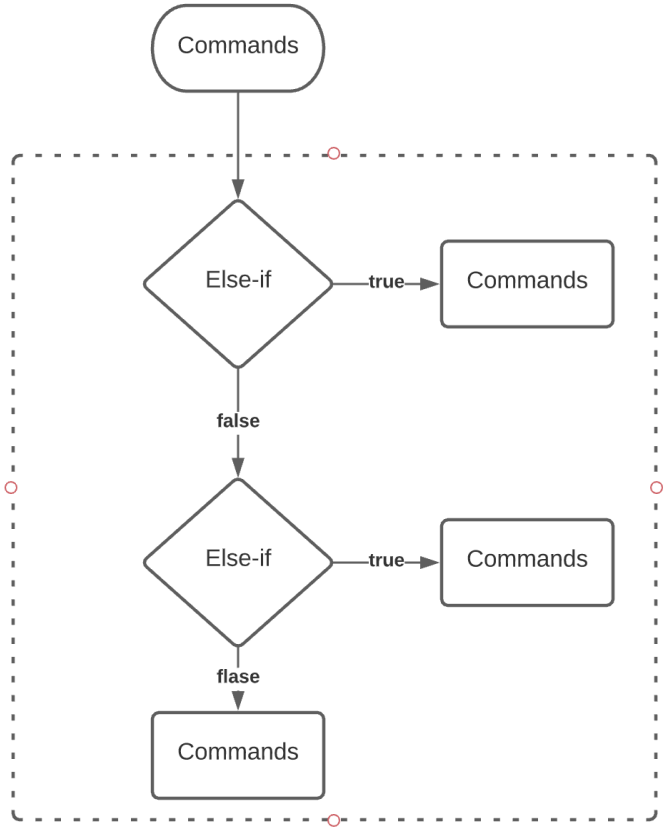
IF condition



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
  commmands3
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

Examples

```
[ -f file ]

[ ! -f 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

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**

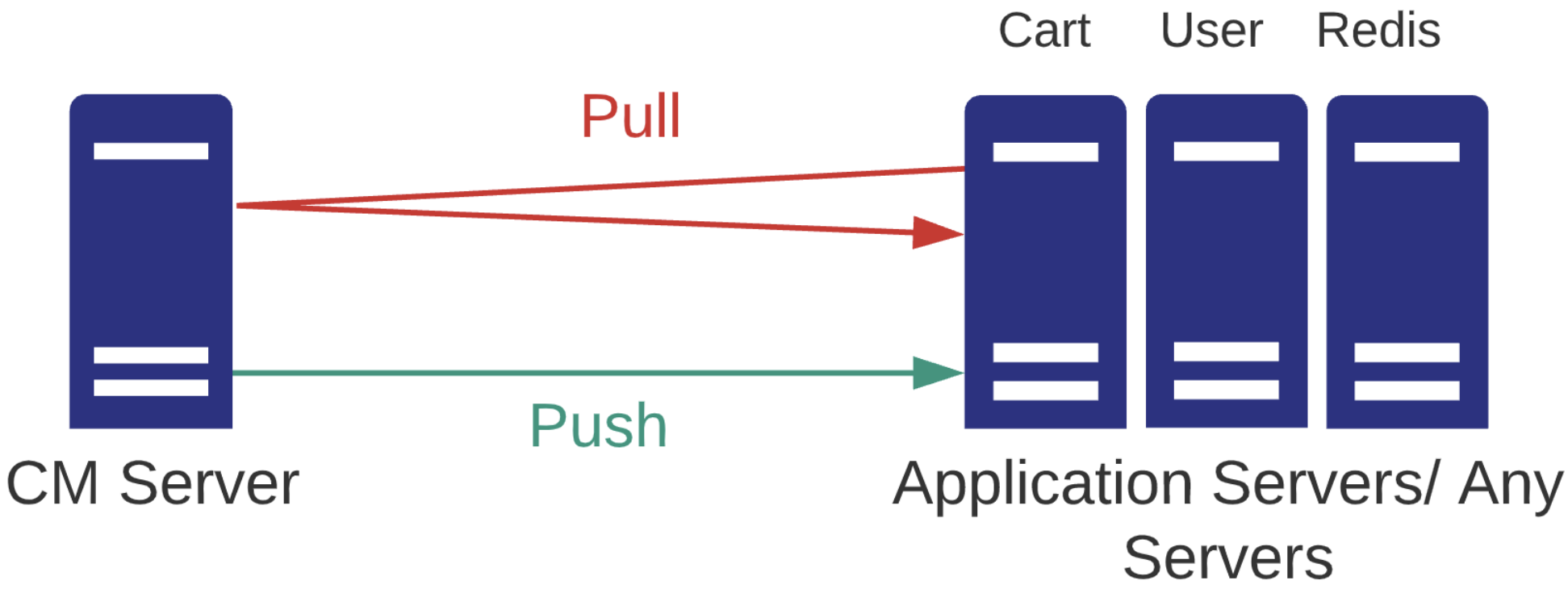
Ansible

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.

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

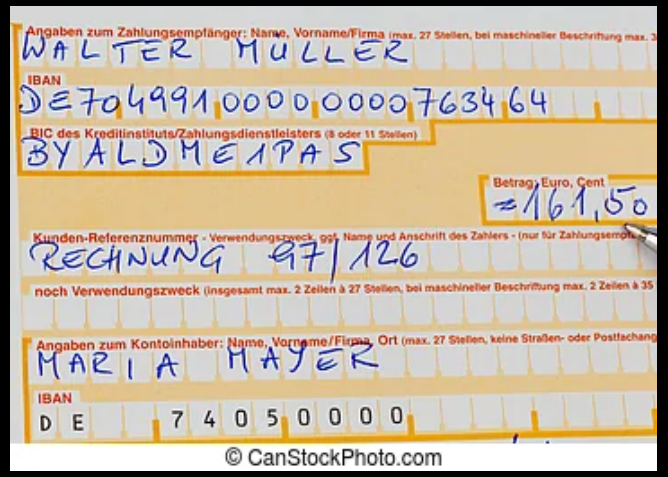
Ansible Modules (Collections)

Problems of Ansible Ad-Hoc

1. Multiple tasks
2. Logical Limtations

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

```
<courseName>DevOps</courseName>
<trainerName>Raghu K</trainerName>
<timings>
   "0600IST",
   "0730IST"
</timings>
<topics>
  <aws>
    "EC2",
    "S3"
  </aws>
  <devops>
    "Ansible"
  </devops>
</topics>
<phoneNumbers>
  <personal>999</personal>
  <mobile>888</mobile>
</phoneNumbers>
```

JSON- Java Script Object Notation

```
{
  "courseName": "DevOps",
  "trainerName": "Raghu K",
  "timings": [
     "0600IST",
     "0730IST"
  ],
  "topics": {
          "aws": [
        "EC2",
        "S3"
      ],
      "devops": ["Ansible", "Shell Scripting"]
    },
  "phoneNumbers": { "personal": 999, "mobile": 888 }
}
```

YAML - Yet Another Markup Lang

```yaml
courseName: "DevOps"
trainerName: "Raghu K"
timings:
  - 0600IST
  - 0730IST
topics:
  aws:
    - EC2
    - S3
  devops: ["Ansible", "Shell Scripting"]
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

Ansible Playbook

- Playbook has multiple plays.
- Playbook file itself is a list.
- Playbook book should have at least one play.

  - Playbook 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.

```
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 key from debug module.

YAML File Extension

.yaml

.yml

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

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
- ***defaults dir from roles***
- Inventory variables

Variable Data Types

```
URL7: vars.google.com
NUMBER: 7
NUM1: 7.7
BOOL1: true
BOOL2: false
BOOL3: yes
BOOL4: no
STRING: "true"
NUMBER2: "7"
```

RoboShop Project Structure

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

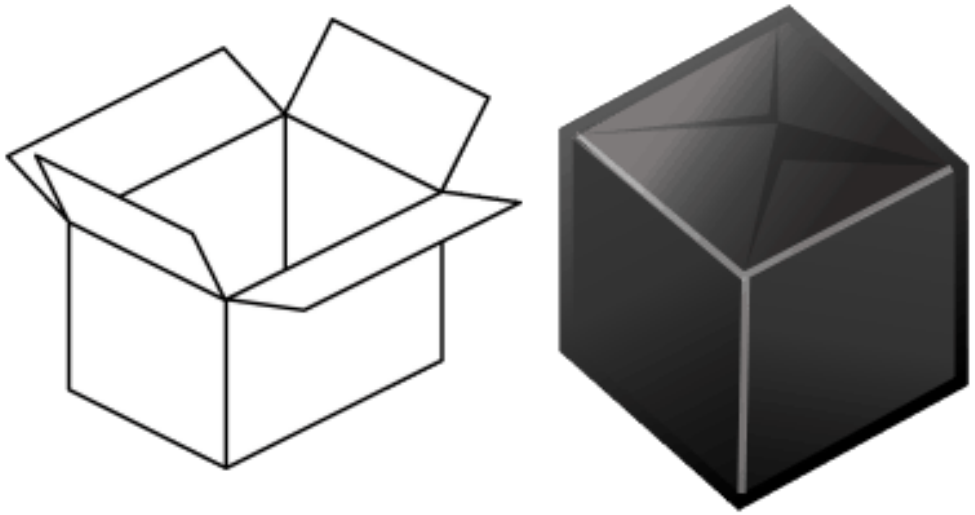
Prometheus

Blackbox Monitoring

Insights cannot be visible

Whitebox Monitoring

Insights are 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

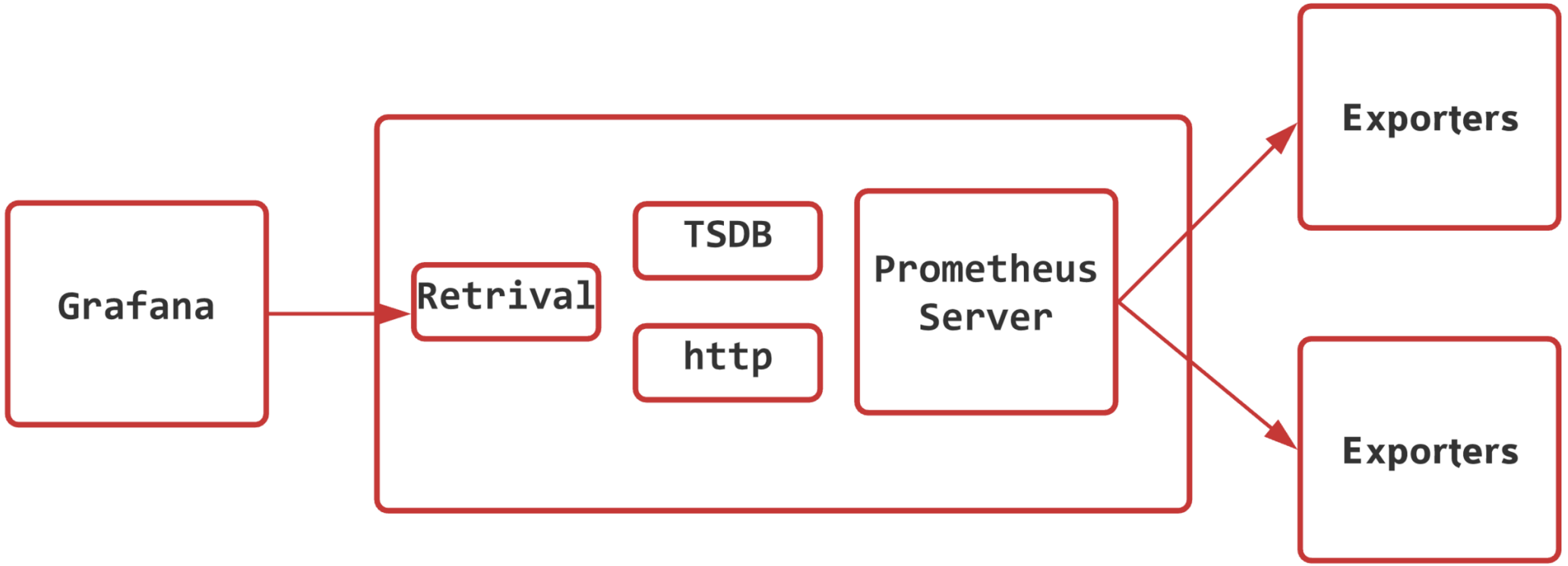
What Prometheus Does no do

- 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 - Exporters



In Prometheus what and which nodes to scrape is given as job configuration

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

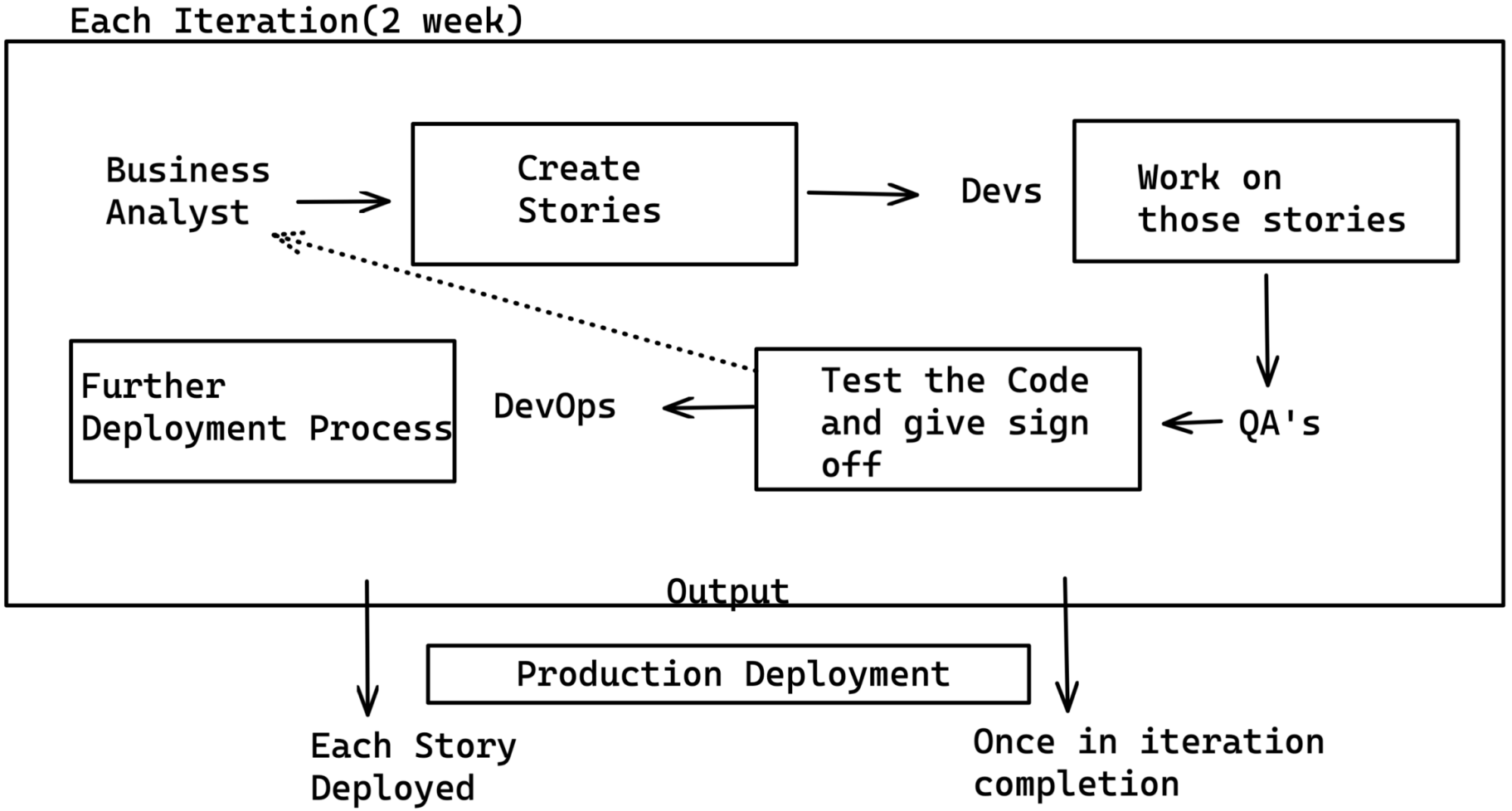
```
groups:
  - name: Alerts
    rules:
      - alert: InstanceDown
        expr: up == 0
        for: 1m
        labels:
          severity: critical
        annotations:
          summary: "Instance Down - [{{ $labels.instance}}]"
          description: "Instance Down - [{{ $labels.instance}}]"
```

Alert Manager

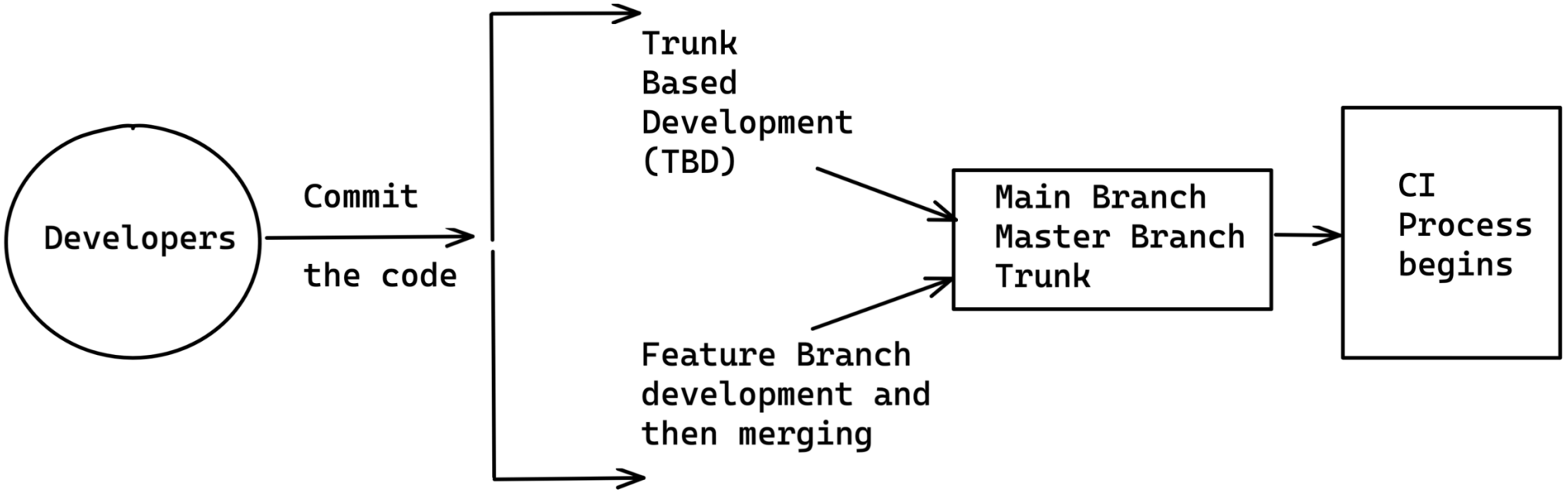
Service Discovery

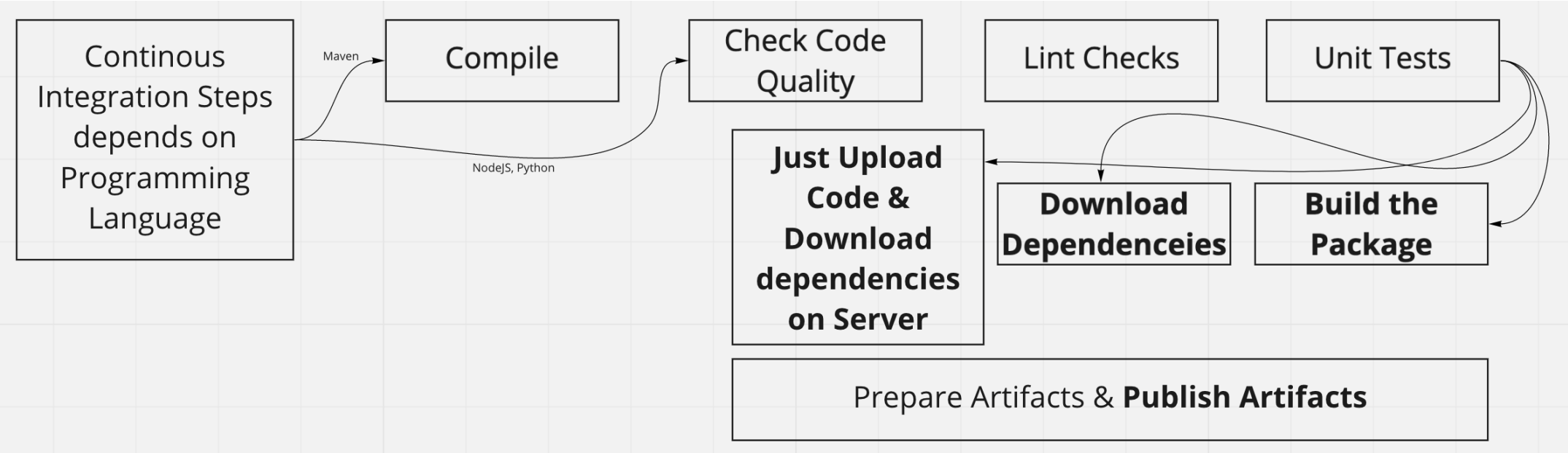
Continous Integration

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.WeekNumber Using GitTags	Using GitTags	Using GitTags
Decides a custom Number	Get Number from CI system	Sprint Number
Pipeline uses number input & Makes Artifact using that number	Pipeline uses number input & Makes Artifact using that number	

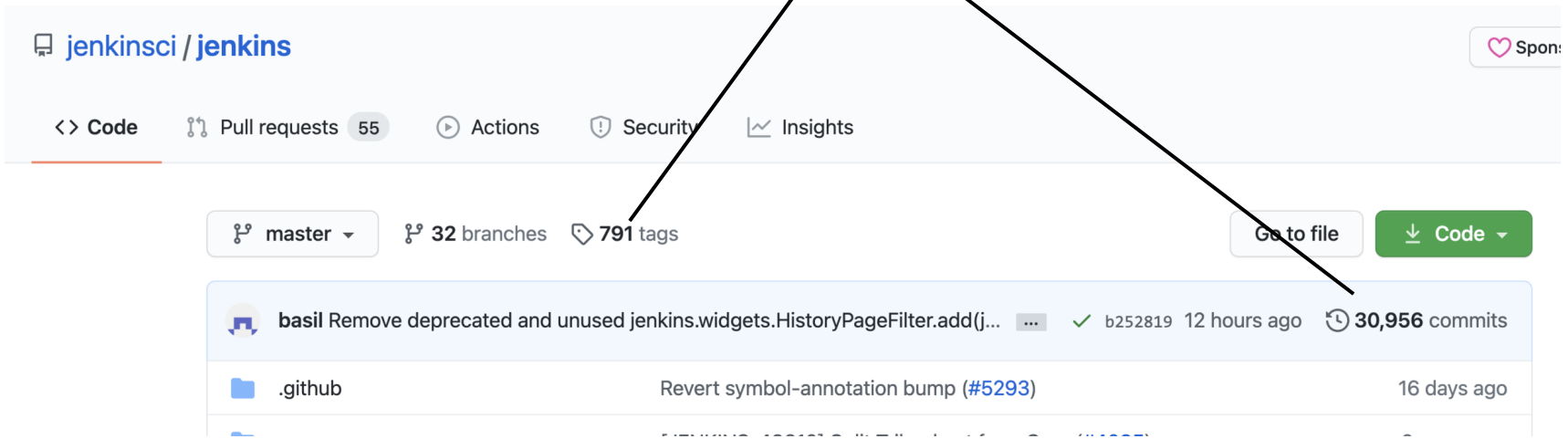
Version Strategy of RoboShop

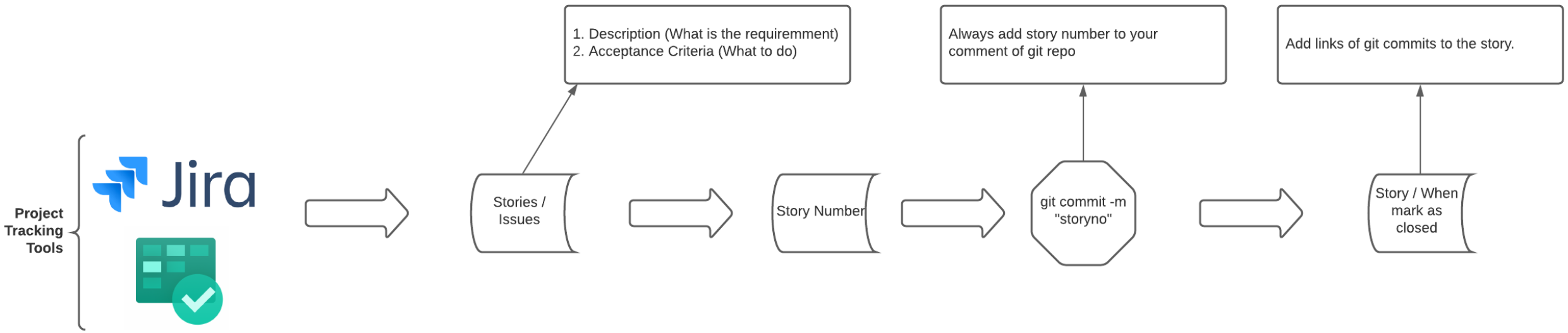
Using **Build Number** from CI for Dev Release

Use Git Tags for PROD Releases

MajorVersion.CI Build Number

MajorVersion.GitTag




Project Tracking Tools
Jira
Stories / Issues
1. Description (What is the requiremment)
2. Acceptance Criteria (What to do)
Story Number
Always add story number to your comment of git repo
git commit -m "storyno"
Add links of git commits to the story.
Story / When mark as closed

Jenkins

Jenkins Used for?

Automation

CI & CD

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.
- **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.

  This pipeline jobs helps you to achieve **GitOps**

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.

Pipeline Jobs

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

Pipeline Jobs are two types

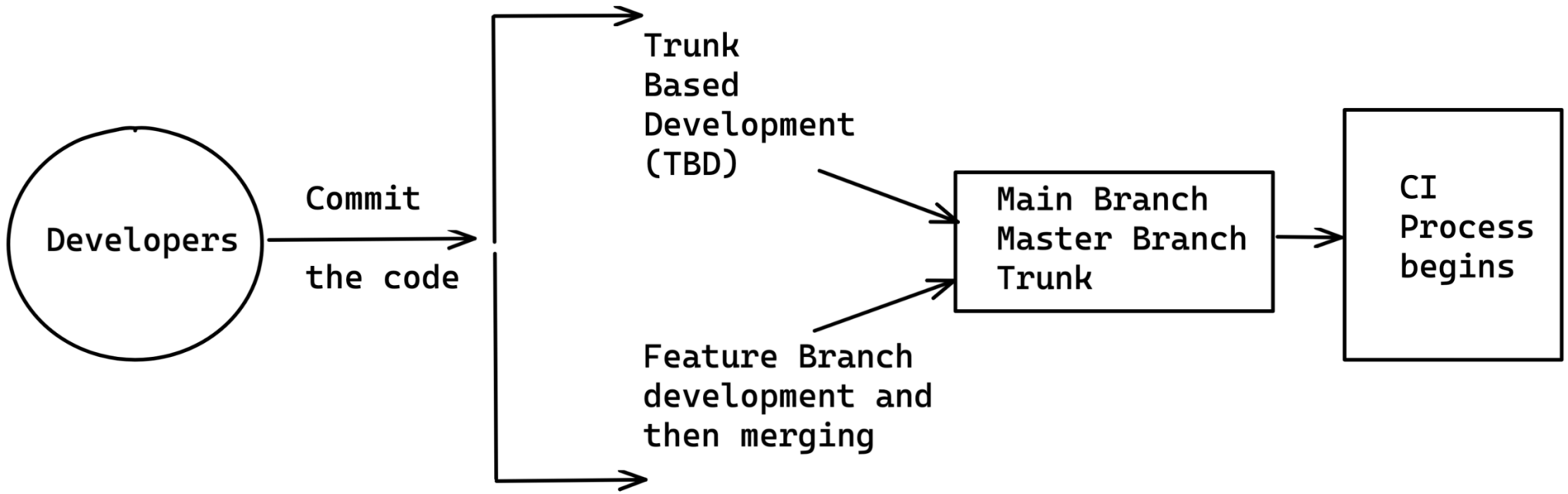
- Scripted Pipeline
- **Declarative (DSL) Pipeline**
- ***Declarative (YAML) Pipelines -> Future***

```
node {
}
```

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

```
pipeline {
}
```

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.


Developers
Commit
the code
Trunk
Based
Development
(TBD)
Feature Branch
development and
then merging
Main Branch
Master Branch
Trunk
CI
Process
begins

Jenkins Architecture

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.

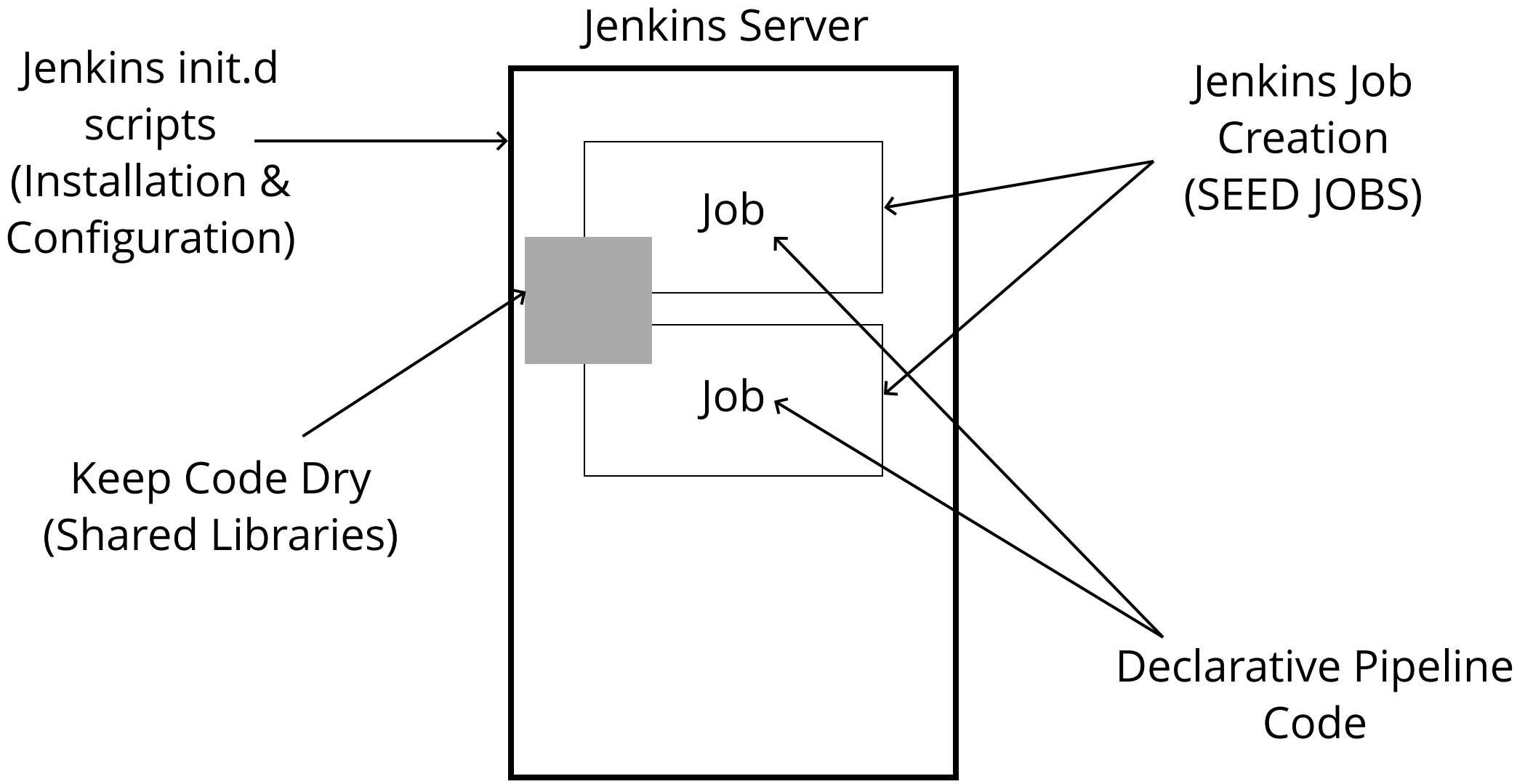
Note: Since we are on lab, keeping cost in mind we may use single node going further.

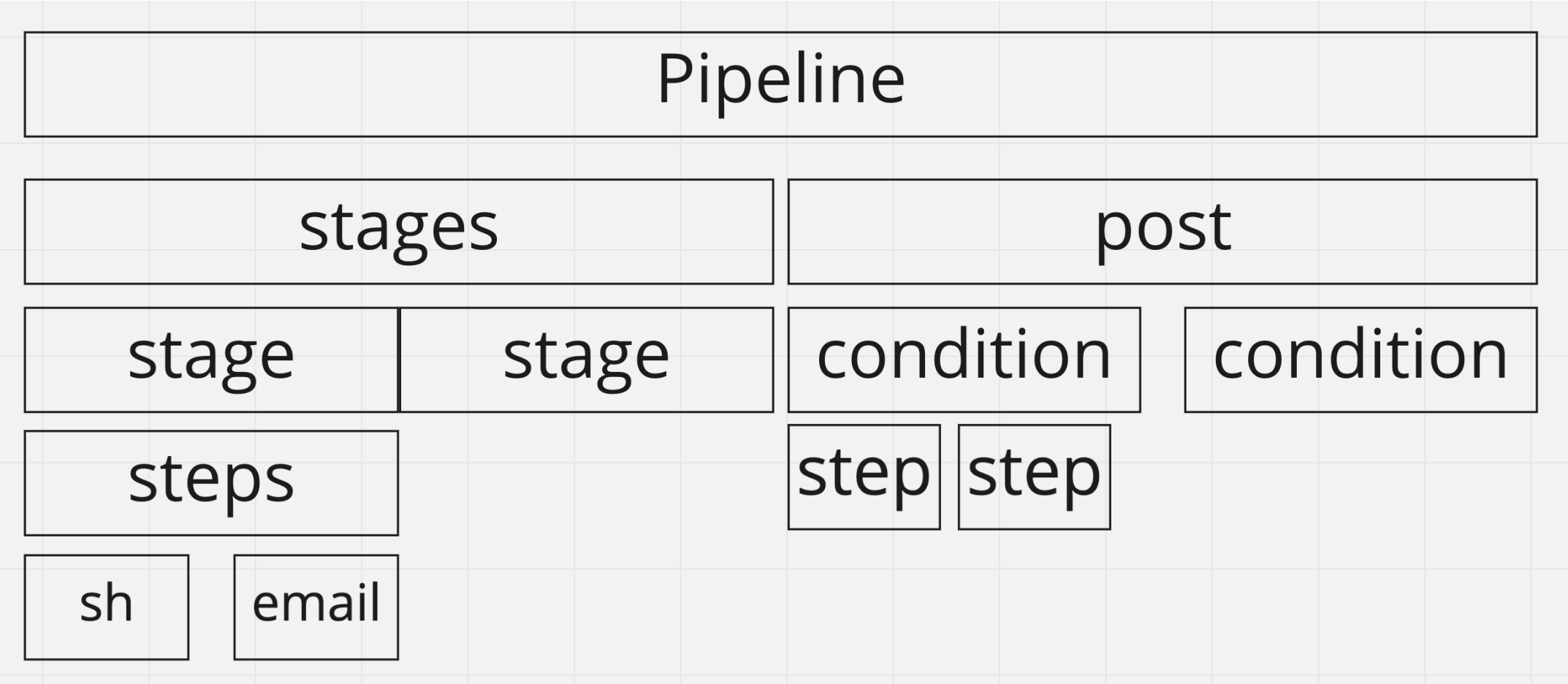
Jenkins Agents

Add node from Jenkins Server

Node reaches Jenkins server and add itself

Jenkins as a Code




Pipeline
stages
post
stage
stage
condition
condition
steps
step
step
sh
email

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. VPN 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

- 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.

Subnets

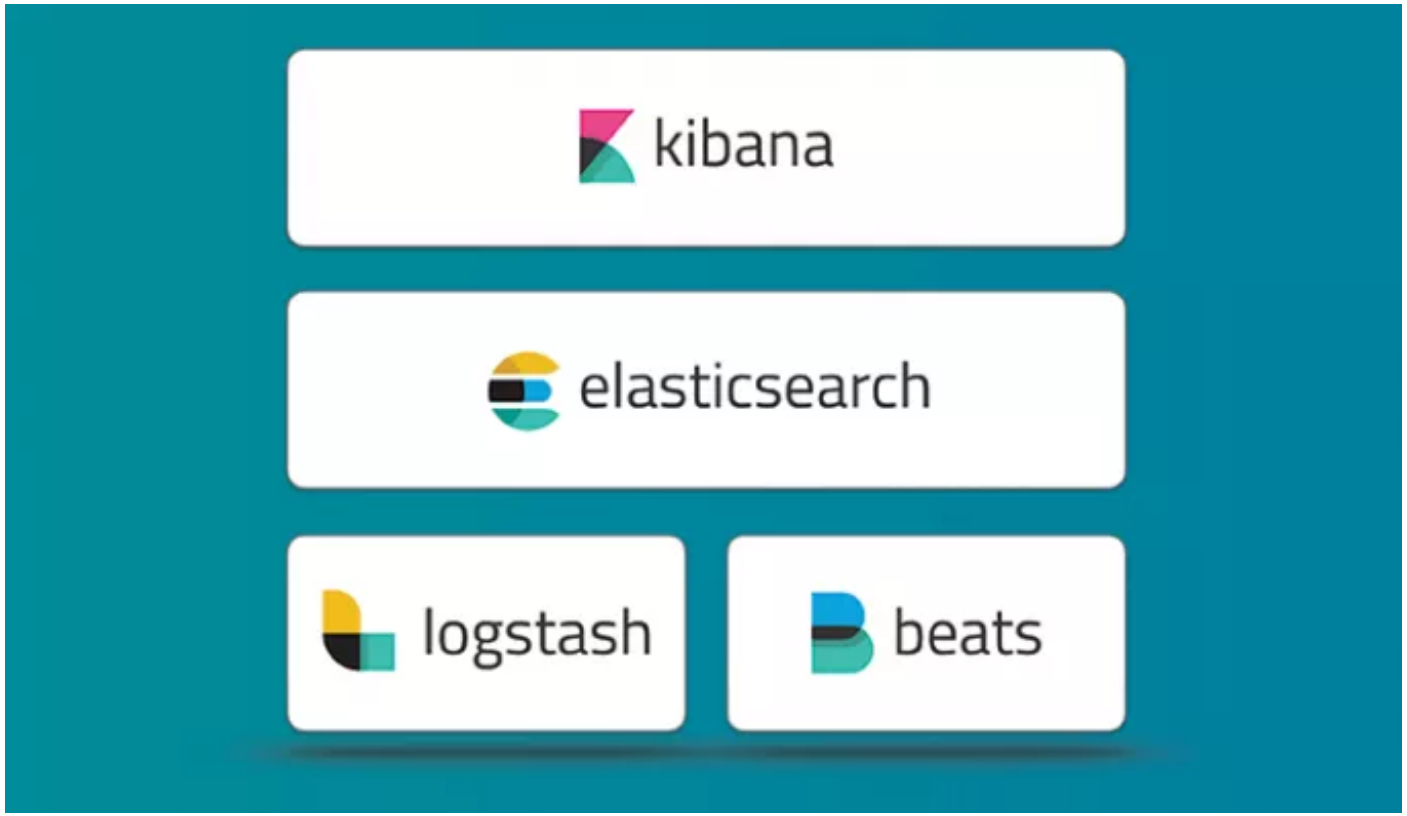
1. Subnets are actual network configs But created for Availability Zones.
2. Once if the subnet is created then only we can create any compute resources like EC2.
3. Certain people create subnets for all the availability zones.
4. Certain people go and create subnets only on desired number of availability 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.

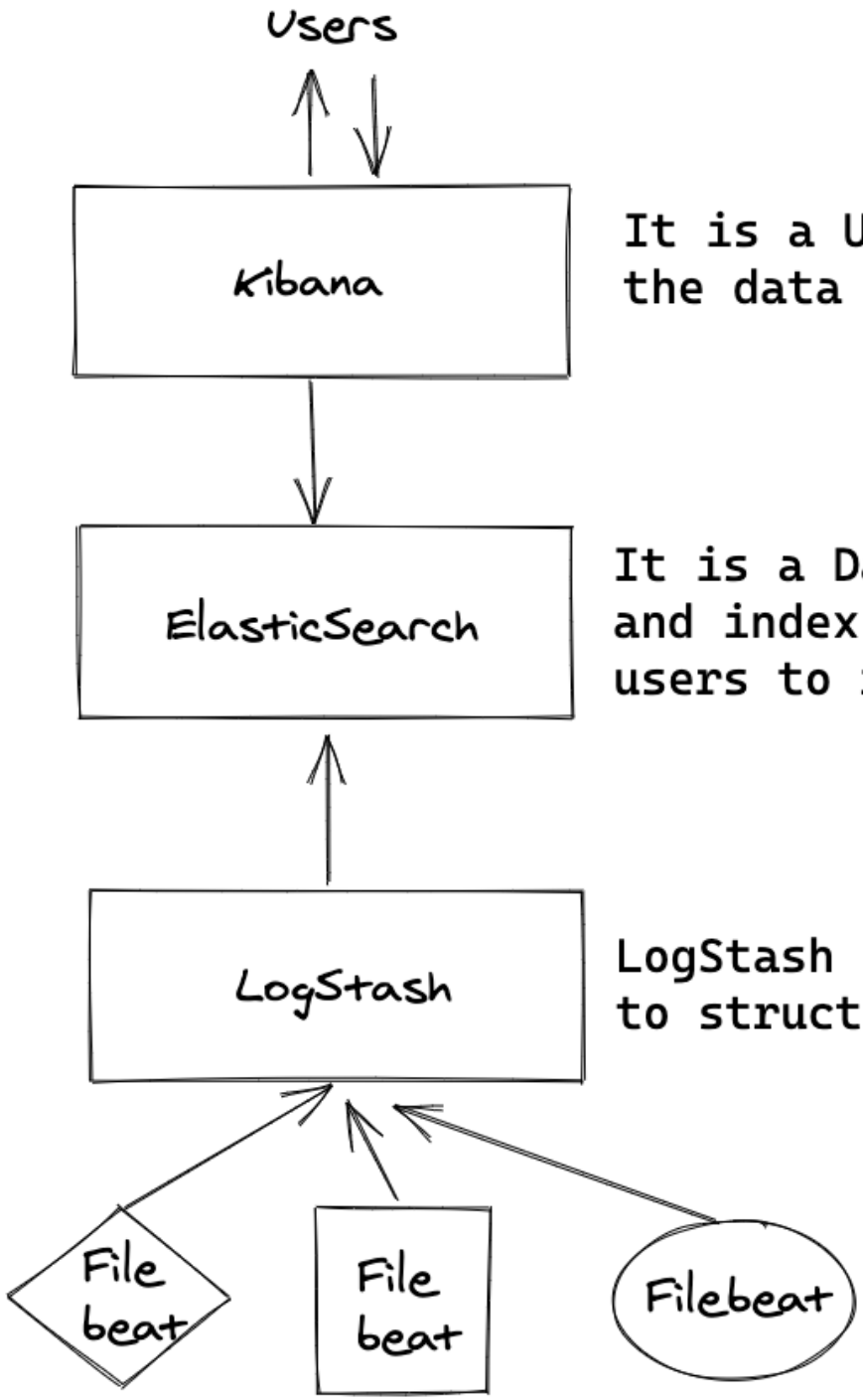
Servers reachability

If all our are 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

ELK





It is a UI for users to search the data in ElasticSearch

It is a Database, Creates and index of words and allows users to retrieve information over HTTP

LogStash converts the unstructured data to structured format.

Filebeat collects the data and it can send it to either logstash or directly to elasticsearch

Document 1

The bright blue butterfly hangs on the breeze.

Document 2

It's best to forget the great sky and to retire from every wind.

Document 3

Under blue sky, in bright sunlight, one need not search around.

Stopword list

a
and
around
every
for
from
in
is
it
not
on
one
the
to
under

Inverted index

ID	Term	Document
1	best	2
2	blue	1, 3
3	bright	1, 3
4	butterfly	1
5	breeze	1
6	forget	2
7	great	2
8	hangs	1
9	need	3
10	retire	2
11	search	3
12	sky	2, 3
13	wind	2

Logs Type

Structured Data

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

Unstructured Data

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

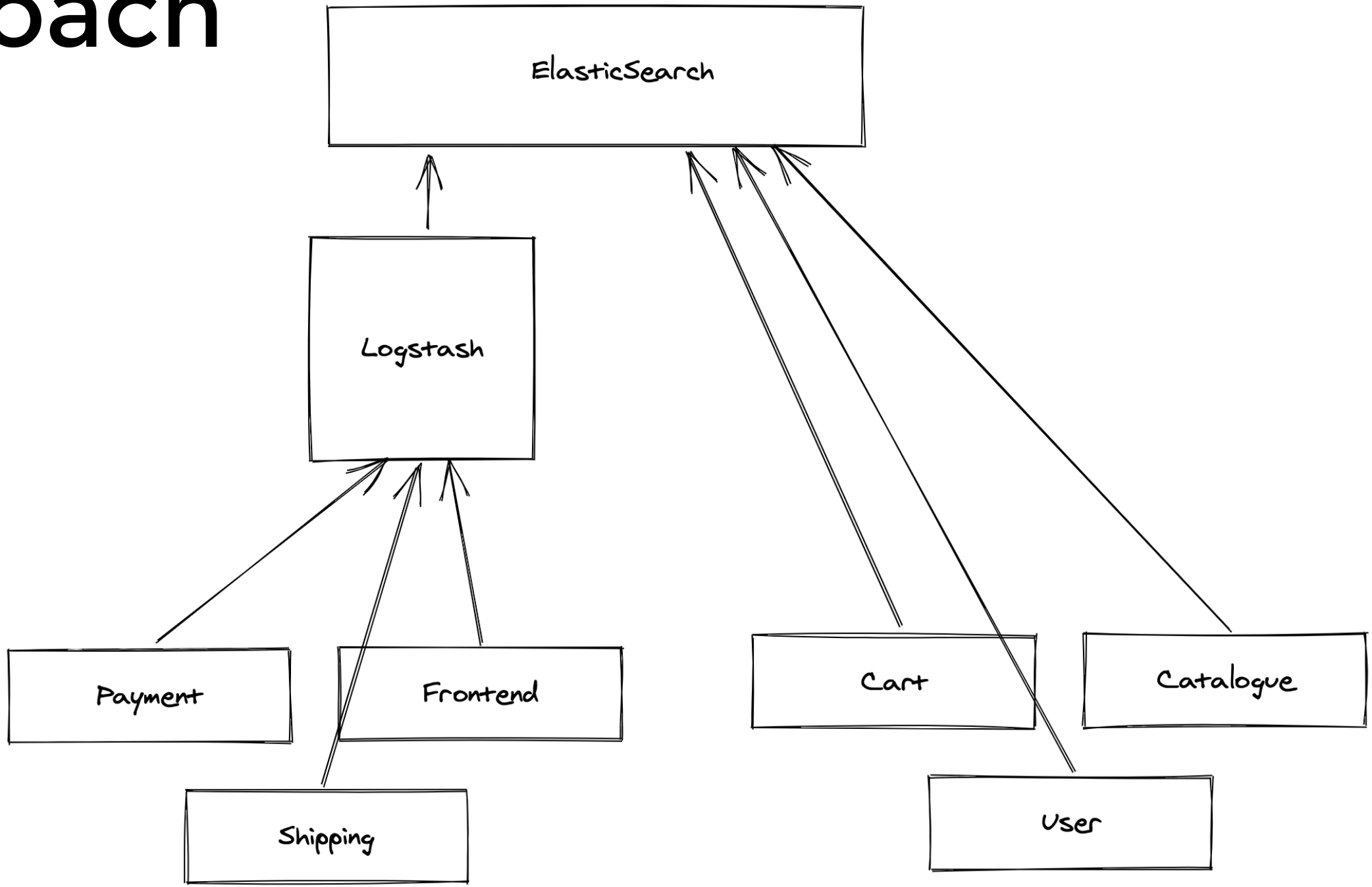
Structured

- 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

Approach

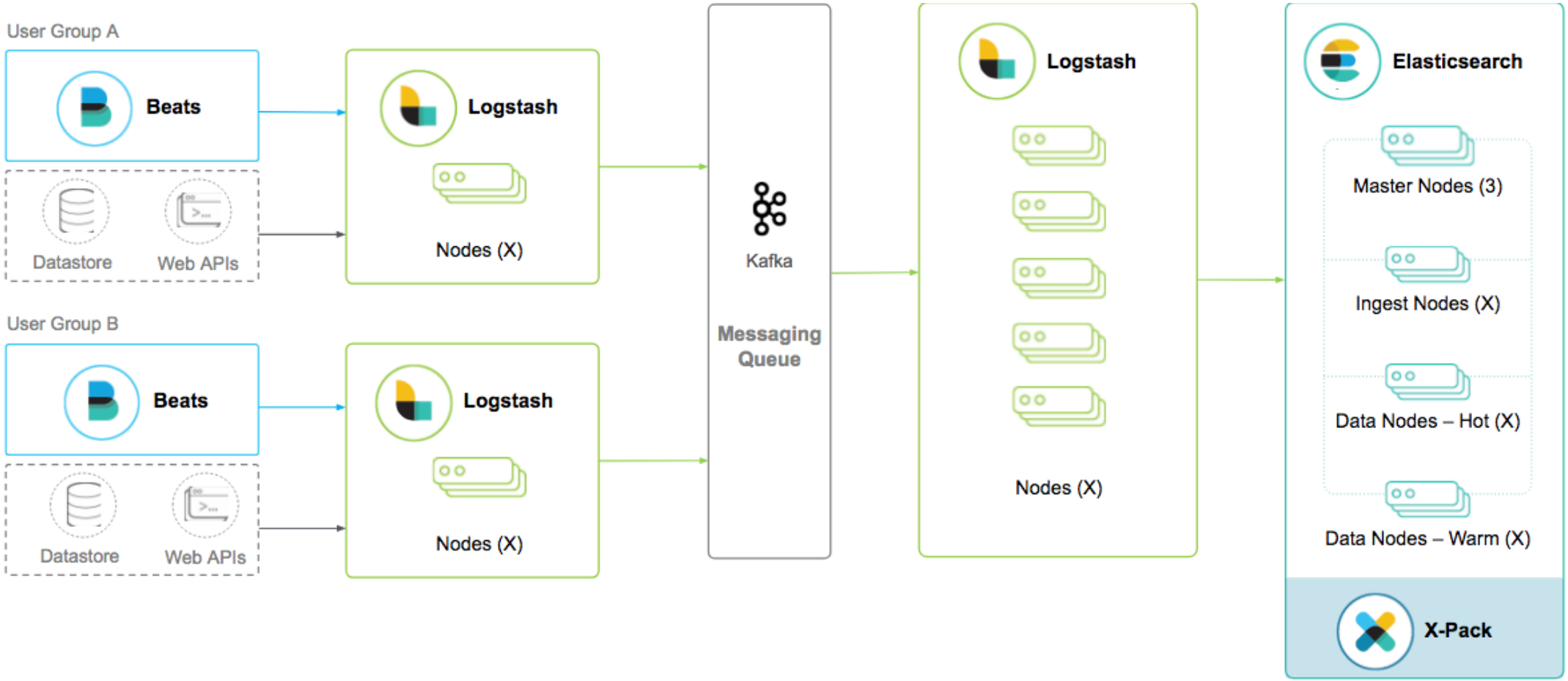

ElasticSearch
Logstash
Payment
Frontend
Shipping
Cart
Catalogue
User

Installation

All in One node

All in Multiple node

Production Grade Cluster



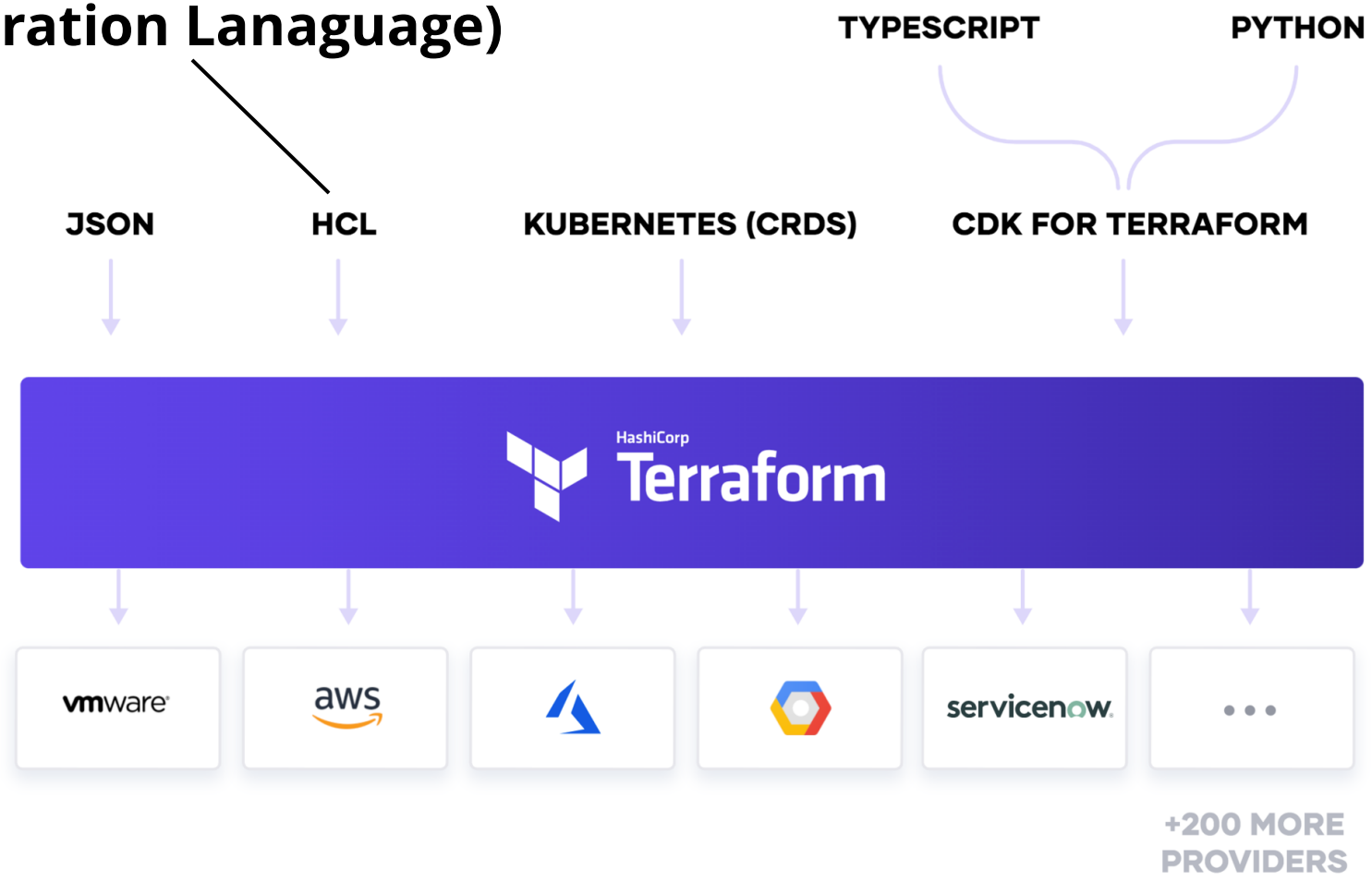
Infrastructure as Code (IaC)

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

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
}

ouptut "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.

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

```
variable "sample" {
  default = "Hello"
}
```

List Variable Type

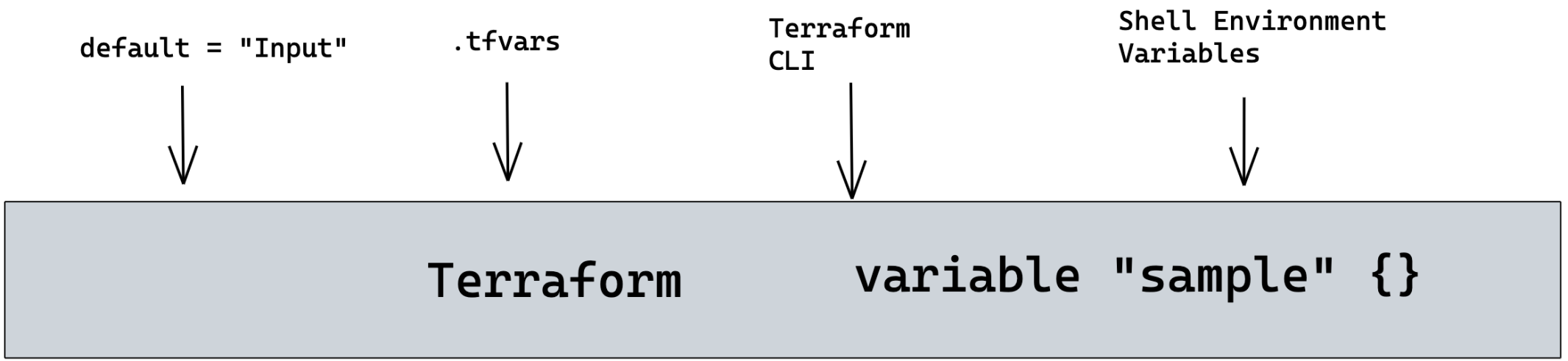
```
variable "sample" {
  default =  [
    "Hello",
    1000,
    true,
    "World"
  ]
}
```

Map Variable Type

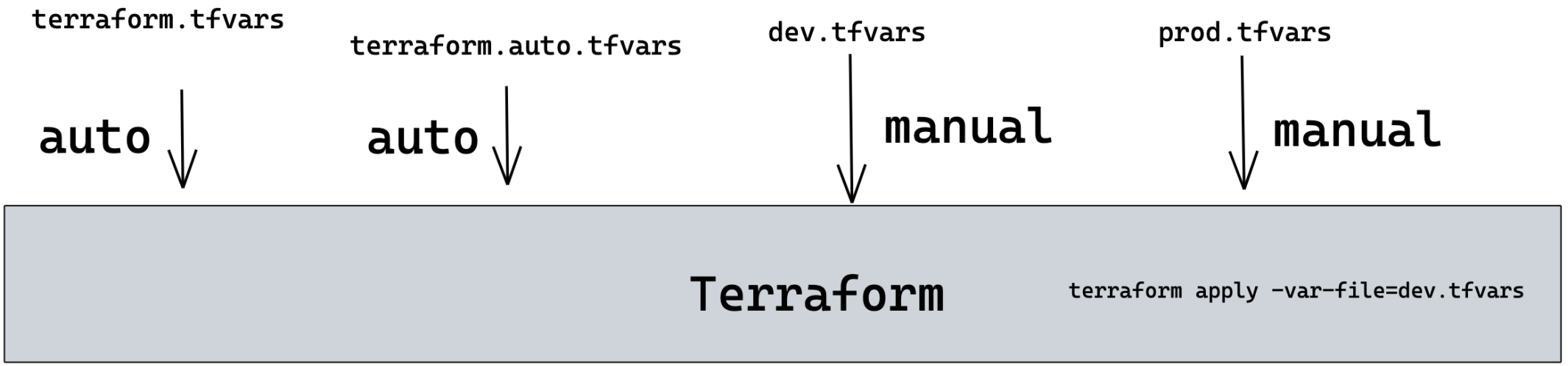
```
variable "sample" {
  default =  {
    string = "Hello",
    number = 100,
    boolean = true
  }
}
```

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

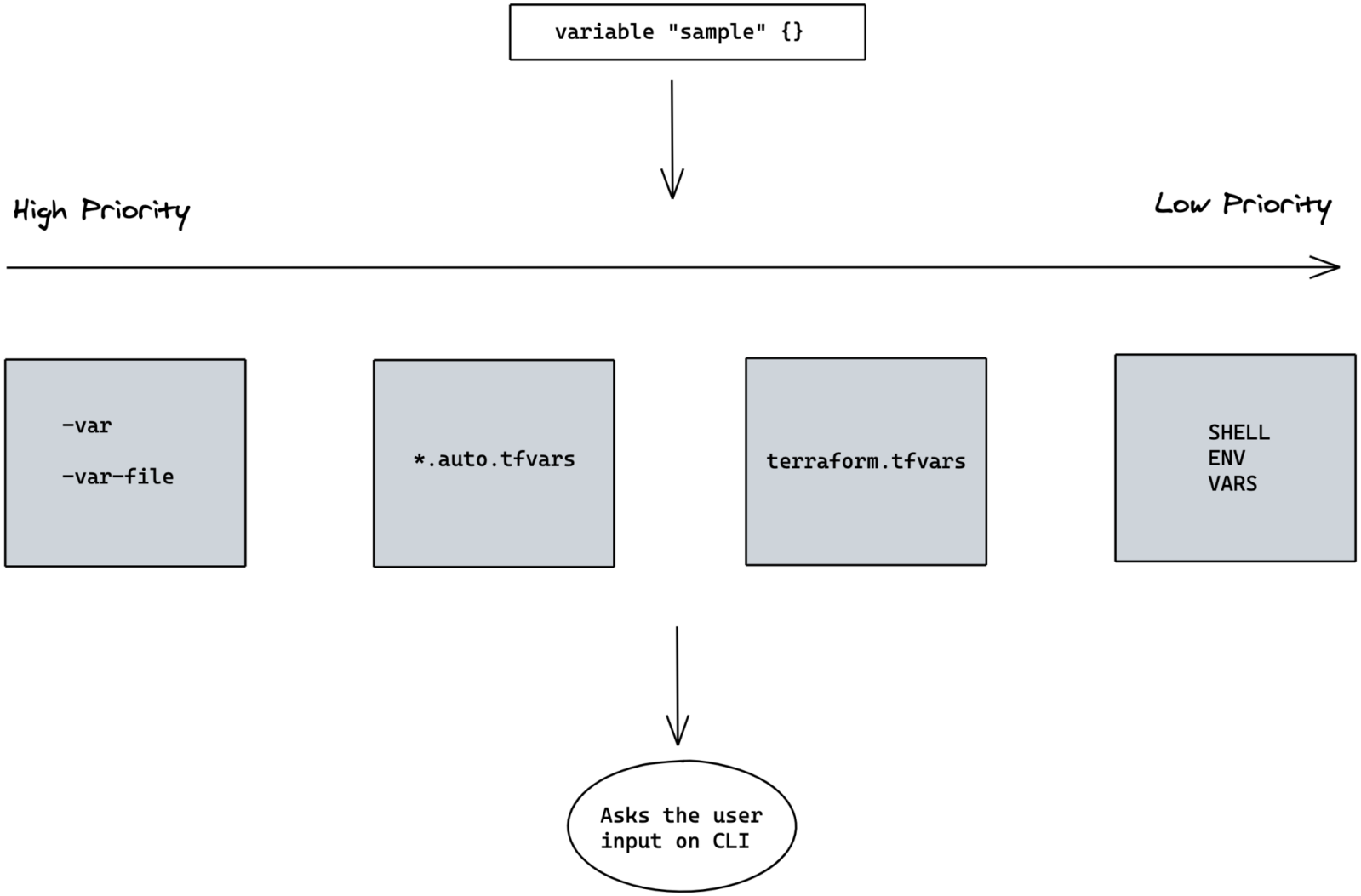
Variables Inputs



Variables TFVARS



Variables Precedence



```
erraform {
 required_providers {
 prod = {
      source  = "hashicorp/aws"
      version = "1.0"
    }
 dev = {
      source  = "hashicorp/aws"
      version = "2.0"
    }

 }


rovider "aws" {}
rovider "azurerm" {}
```

Provider

- A sample provider is what you see on here.
- You can deal with multiple providers at the same time.
- From v0.12 , the Provider declaration is not mandatory. Based on the resources it can pick the provider.
- 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 inputs, 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**. **publc_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

```
module "sample" {
  source = "./ec2"
}

module "consul" {
  source =
"github.com/hashicorp/example"
}
```

- 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.**
- In Terraform 0.13 version they introduced **depends_on & count** keywords for module as well.
- You cannot share the data from one module to another module directly. But can be done only through **Root Module.**

Terraform Loops

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

- 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

```
resource "aws_instance" "sample" {
  count         = condition ? true_val : false_val
  ami           = var.AMI[count.index]
  instance_type = var.INSTANCE_TYPE
}
# count         = var.CREATE ? 1 : 0
# count         = var.ENV == "PROD" ? 1 : 0

variable "CREATE" {
  default = true
}
```

- 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

```
variable "image_id" {
  type        = string
  description = "The id of the machine image (AMI) to use for the server."

  validation {
    condition     = length(var.image_id) > 4 && substr(var.image_id, 0, 4) == "ami-"
    error_message = "The image_id value must be a valid AMI id, starting with \"ami-\"."
  }
}
```

- 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.

Terraform Datasources

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

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.

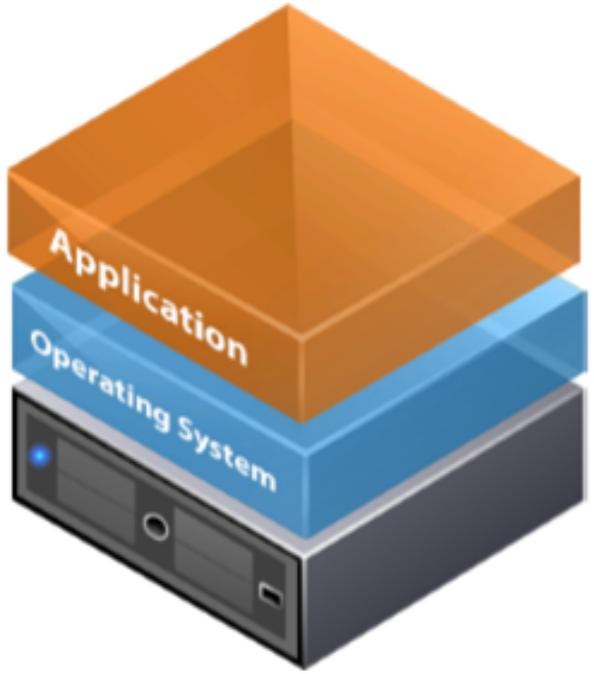
Project Code Structure

Terraform In-Built Functions

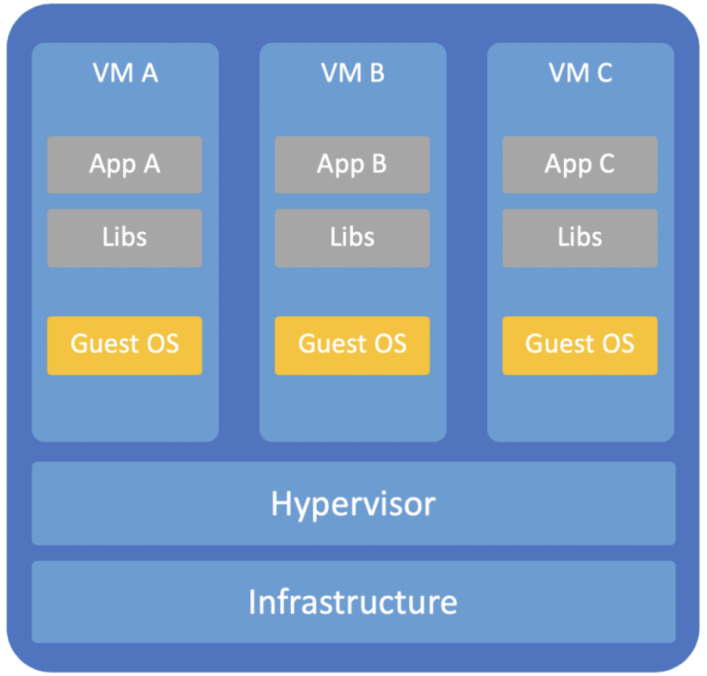
- Terraform have inbuilt functions to manage and transform the data.
- It has numerous functions available.
- A user cannot define their own function in terraform.

Containers

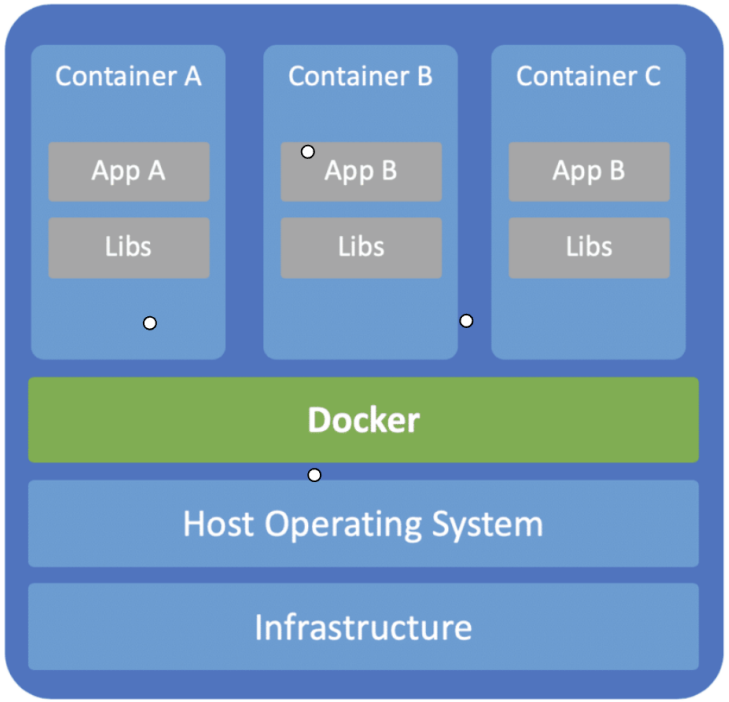
Compute Evolution



Virtual Machines



Container



- 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.

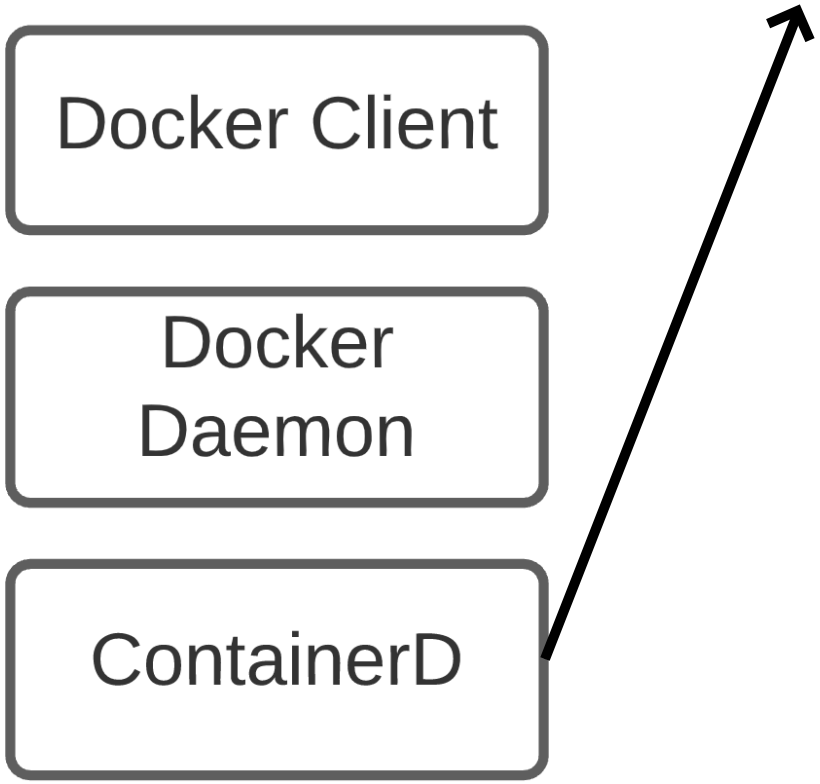
Best UX (User Experience)

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



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 of 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

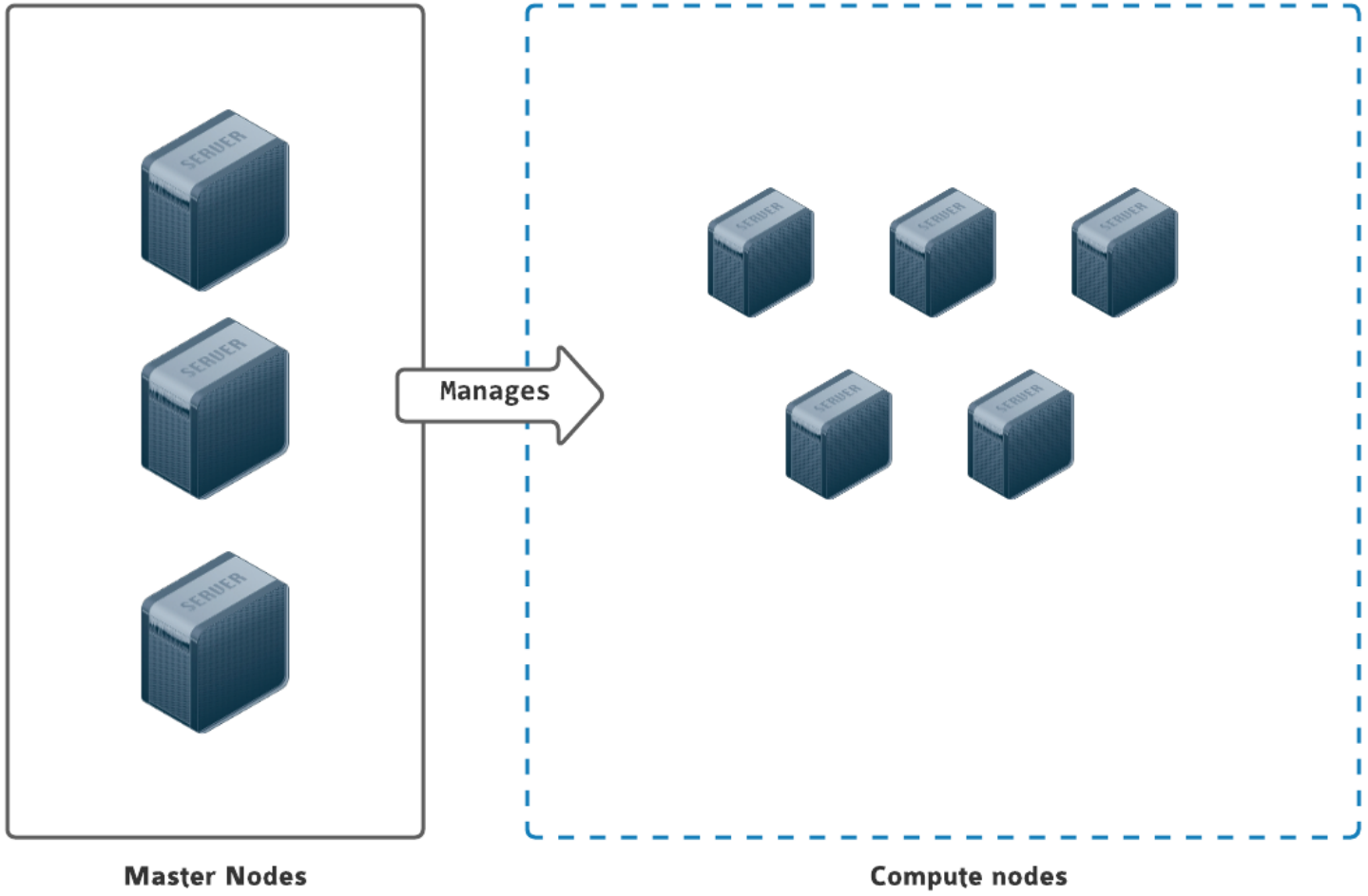
Orchestrator

- Kubernetes
- DCOS
- Docker Swarm

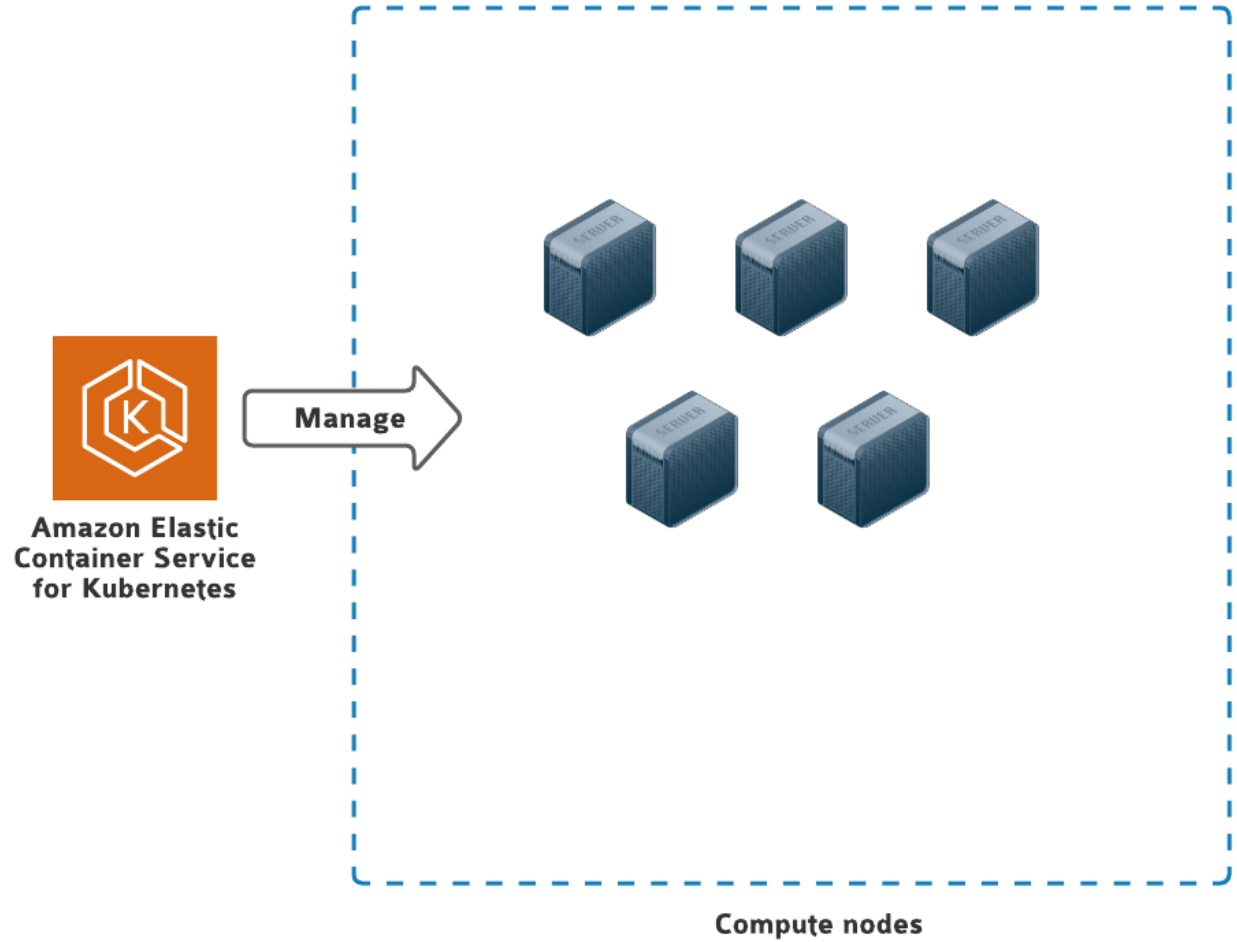
Why Kubernetes?

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

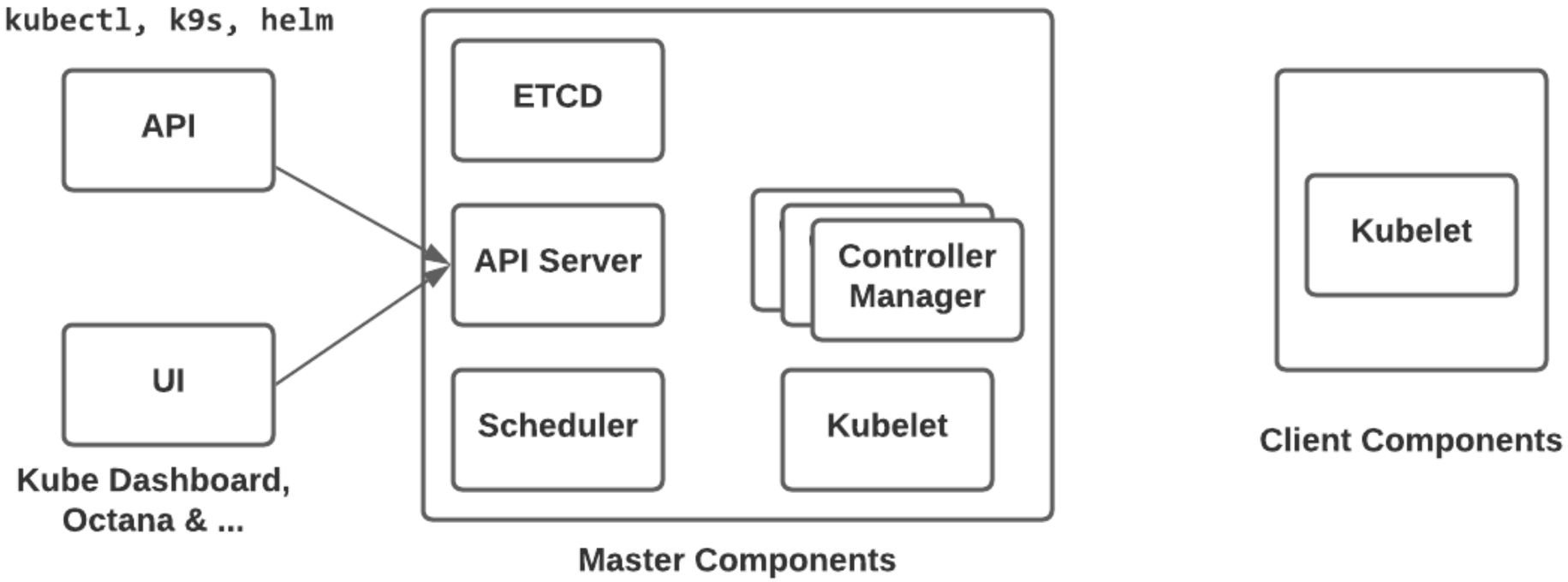
Kubernetes Architecture



Kubernetes Architecture



Kubernetes Components



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

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: sample1
spec:
  containers:
    - name: nginx
      image: nginx
```

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 Pod 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 **PersitentVolume**, **PersistentVolumeClaim** & **StorageClass**

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.