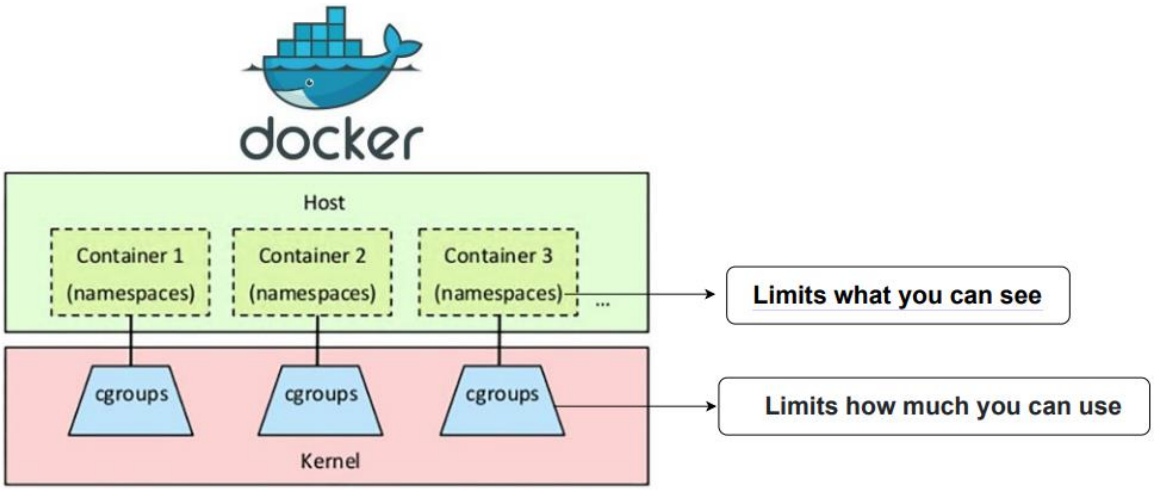
DevOps Concerns

- Containerization ----> Docker , containerd
- Log management ----> ELK stack(elasticsearch)
- CI/CD -----> jenkins , gitlab-ci
- Monitoring -----> prometheus
- Automation -----> ansible , poppet , chef
- Optimization -----> reduce container size , …
- Security -----> wazuh , IPS/IDS
- Linux and service administration ----> bash script , …
- Orchestration -----> swarm , kubernetes
- Code registry -----> gitlab
- Repository -----> nexus , harbor
- Code quality check (vulnerability check,clean code check,stress test, …) ----> sonarqube

Namespace and cgroup



https://github.com/ehsanDadashi/NameSpace_and_cgroup.git

IP tables

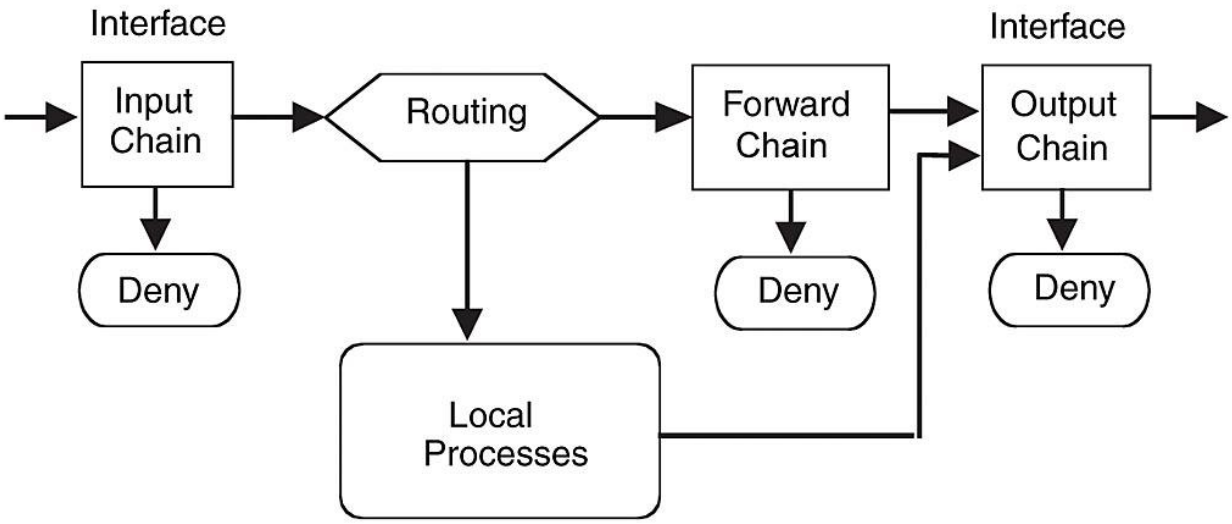
نام قدیمی iptables برابر ipchain بوده است .

Table :

در نرم افزار iptables به ۴ قابلیت ذیل جدول یا table می گویند .

Chain :

به هر قانونی که در این نرم افزار نوشته می شود یک chain می گویند .
جدول default مربوط به iptables برابر filter می باشد .



دارای ۴ جدول پیشفرض است :

- Filter : عملکردی مثل firewall
- Mangle : کیفیت سرویس مثلا محدود کردن پهنای باند
- NAT :
- Raw

Iptables -t [table] CMD [chain] [rule-spec | num] -j [action]

در این دستور CMDمی تواند انواع ذیل را داشته باشد :

-A : append command to chain (دستور در انتهای زنجیر قرار می گیرد)
-I : Insert rules in chain (دستور در ابتدای زنجیر قرار می گیرد)
-D : delete a rule from chain (حذف دستور)
-R : Replace a rule from chain (جایگزین کردن دستور)

در قسمت chainمی توان ۳ مود را در نظر گرفت :

Chain : (INPUT | FORWARD | OUTPUT)

در قسمت (rule-spec|num) که filtering specification می باشد سوئیچ های ذیل وجود دارد :

-p <protocol-type>
-s <source ip address>
-d <destination ip address>
-i <input interface name>
-o <output interface name>
-sport <source port number>
-dport <destination port number>

در قسمت actionنوع رفتار firewall مشخص می گردد :

Action (REJECT | DROP | ACCEPT)

Sample :

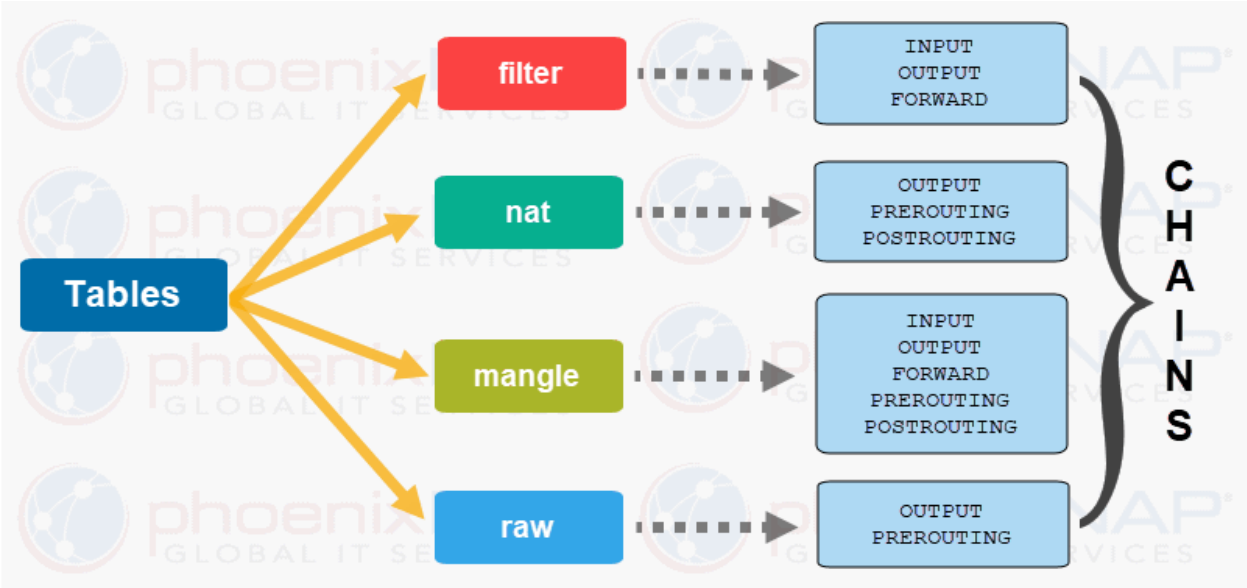
- مسدود کردن دسترسی به یک IP خاص:
  iptables -A INPUT -s 192.168.1.100 -j DROP

- مجاز کردن ترافیک HTTP (پورت 80)

  iptables -A INPUT -p tcp --dport 80 -j ACCEPT

- مسدود کردن تمام ترافیک ورودی به جز SSH

  ابتدا تمام ترافیک ورودی مسدود می گردد ----- iptables -P INPUT DROP
  در واقع نو رویکرد (رفتار پیش فرض) را DROP می کند . این رفتار برای chain ها
  در نظر گرفته می شود . در واقع تمام ورودی ها drop شود مگر پایینی
  حال پورت ۲۲ مجاز می ---- iptables -A INPUT -p tcp --dport 22 -j ACCEPT
  گردد .



LINUX

- ACL:

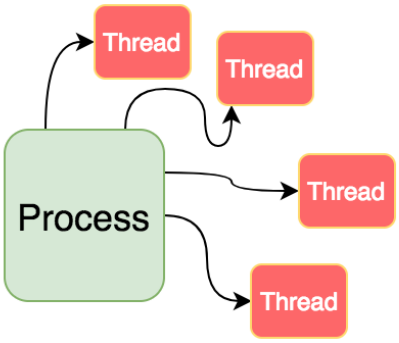
با استفاده از دستور chmod می توان سطح دسترسی فایل را برای خود owner و گروه owner و گروه other تنظیم کرد اما نمی توان به کاربر یا گروه خاصی دسترسی مشخصی داد و این کار با استفاد از access list command ها که setfacl و getfacl می باشند امکان پذیر است .
برای تنظیم سطوح دسترسی به فایل ها استفاده می گردد .
Setfacl , getfacl

setfacl -option file_owner:file_permission filename

Sample :

setfacl -m u:kali:rw test.txt

- Process vs thread :

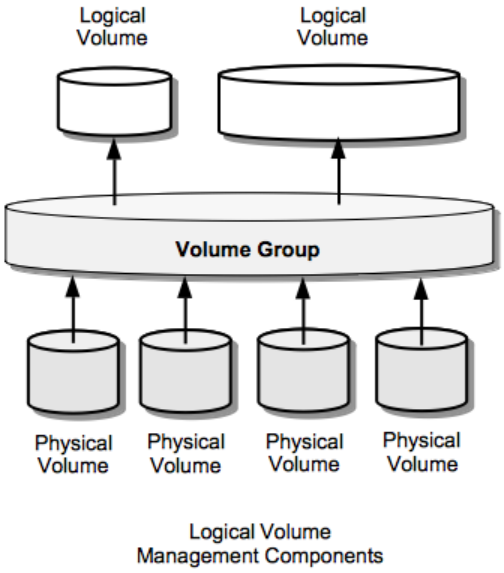


پروسس : هر App جهت اجرا به یک پروسس نیاز دارد .
ترد : هر پروسس یک یا چند ترد برای اجرا دارد (ترد بخشی از یک پروسس است).

- Add disk to linux :

LVM logic ----> first pv(physical volume) then vg(volume group)
and finally lv(logical volume)

ابتدا با Command مربوط به PV یک physical volume به سرور اضافه می کنیم
این کامند برابر pvcreate می باشد و سپس آن را به گروه volume ها اضافه
می کنیم (vgextend) و در نهایت آن را به LVM مربوطه اضافه می کنیم (lvextend)
در نهایت فایل سیستم را resize می کنیم (resize2fs)



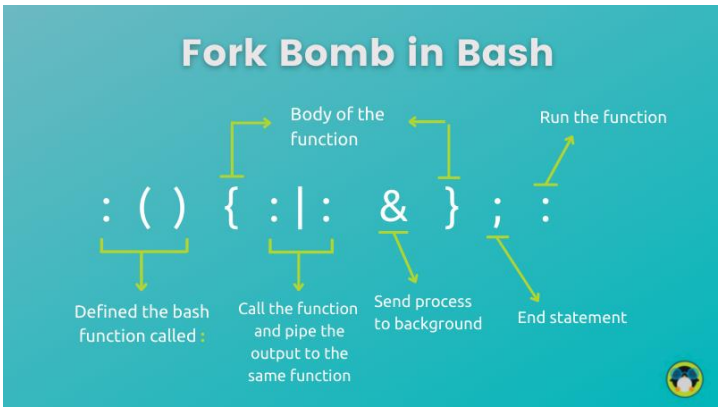
- File attribute :

نوعی metadata است که رفتار file system را برای فایل تعریف می کند (دسترسی و modify کردن فایل را توضیح می دهد) . اگر فلگ i با دستور ذیل به آن اضافه شود حتی کاربر root هم نمی تواند آن را حذف کند .

Flag i : read only
sudo chattr +i filename

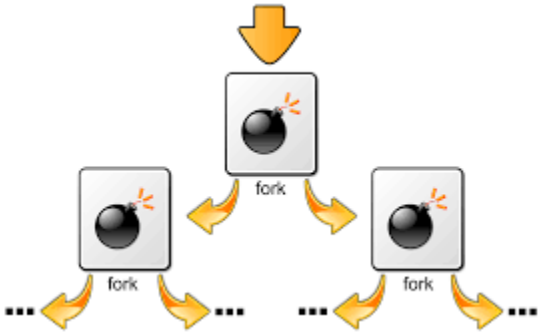
- FORK BOMB

یک process که بارها و بارها خودش را تکرار می کند تا تمام منابع سیستم را مصرف کند .
Create Fork bomb command linux --> :() { : | :& };:



میتونستیم به شکل ذیل هم بنویسیم :

bomb() { bomb|bomb& }; bomb



- Zombie process

فرآیندی است که زمان اجرا آن به پایان رسیده اما هنوز از جدول فرآیندها حذف نشده است . این وضعیت زمانی رخ می دهد که فرآیند فرزند به پایان رسیده و منتظر است تا فرآیند والد وضعیت خروج آن را دریافت کند .

فرآیند زامبی به خودی خود منابع سیستم را مصرف نمی کند اما اگر تعداد فرآیندهای زامبی زیاد باشد می تواند باعث پر شدن جدول فرآیند ها گردد .

ps -el | grep Z ----> show zombie process

- Systemd components:

یکی از اصلی‌ترین سیستم‌های init در بسیاری از توزیع‌های مدرن لینوکس است که فرآیند بوت و مدیریت سرویس‌ها را برعهده دارد. این سیستم شامل مجموعه‌ای از کامپوننت‌هاست که هر کدام نقش خاصی در مدیریت سیستم دارند.

1. Systemctl

یک ابزار خط فرمان برای کنترل و مدیریت سرویس‌ها و واحدهای systemd. با استفاده از این ابزار می‌توان سرویس‌ها را شروع (start)، متوقف (stop)، راه‌اندازی مجدد (restart) و وضعیت آنها را بررسی کرد.

2. journalctl

مشاهده، ابزار خط فرمان برای دسترسی به گزارش‌های ثبت شده می باشد
فیلتر کردن و جستجوی لاگ‌ها را فراهم می‌کند
و systemd-networkd و systemd-timesyncd و ...

Deployment Methods

- recreate deployment

  در این روش ابتدا نسخه قدیمی به طور کامل متوقف می شود و سپس نسخه جدید مستقر می گردد

- Rolling Deployment

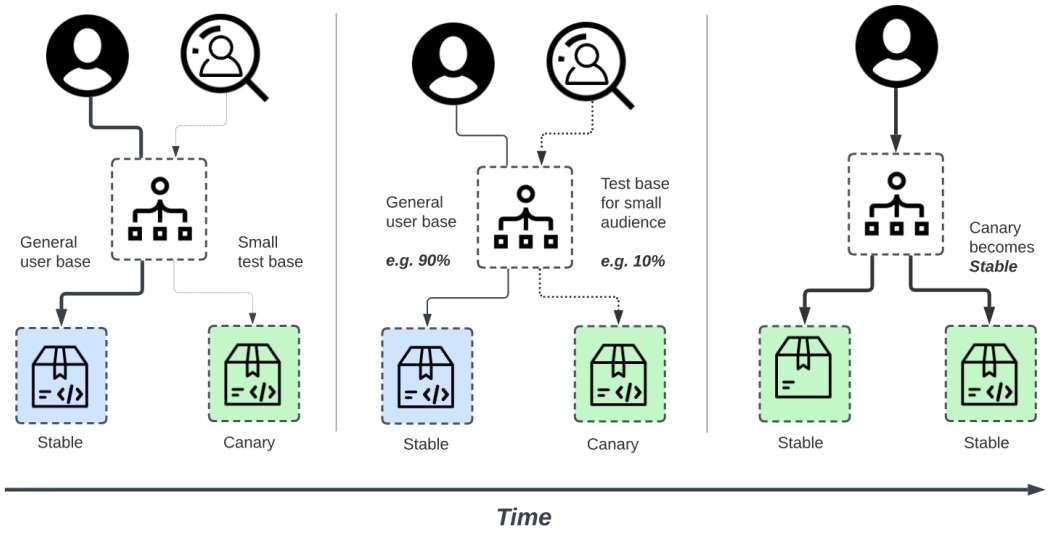
  نسخه های جدید به تدریج جایگزین نسخه های قدیمی می شوند .
  مزایا : حداقل وقفه در سرویس دهی
  معایب : ممکن است برای مدت کوتاهی هر دو نسخه جدید و قدیم در دسترس باشند که می تواند مشکل سازگاری ایجاد کند .

- blue -green deployment

  هر دو محیط قدیمی و جدید در آن واحد وجود دارد و ترافیک بین آن ها تغییر می کند .

مزایا : هرگز end user متوجه downtime نخواهد شد
کاهش ریسک قبل از عملیاتی سازی
معایب : نیاز به دو محیط مشابه و هزینه بالاتر

- canary deployment :
  نسخه جدید ابتدا برای تعداد کمی از کاربران publish می گردد پس از تایید نهایی برای همه مستقر خواهد شد .



مزایا : امکان مقایسه و انتخاب بهترین نسخه
معایب : نیاز به تحلیل و مدیریت پیچیده

Bash script

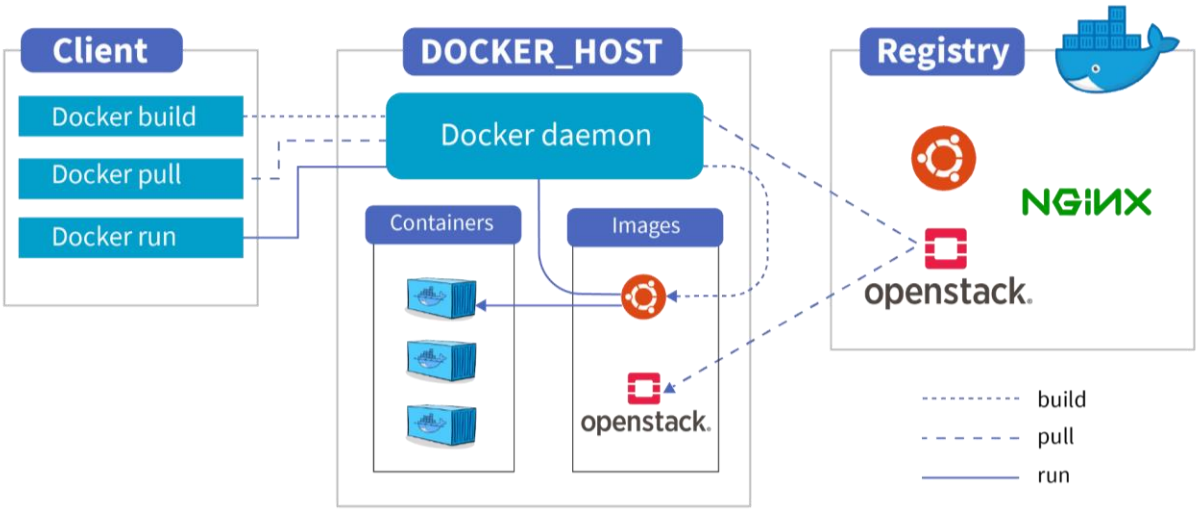
- echo $? ----> نتیجه آخرین دستور را در خود ذخیره می کند اگر برابر ۰ باشد موفق و در غیر این صورت عددی غیر از ۰ خواهد بود .
- X = $(echo $?) ---> ذخیره می کند x نتیجه دستور داخل پرانتز را در متغییر .

```
#!/bin/bash                                  ---> shebang
# Function to calculate factorial
factorial() {                                ---> define function
 number=$1
 result=1
 for (( i=1; i<=number; i++ ))
 do
   result=$((result * i))
 done
 echo $result
}

# Main script starts here
echo "Enter a number:"
read number

# Check if the input is a non-negative integer
```

```
if [[ $number =~ ^[0-9]+$ ]]; then
  result=$(factorial $number)
  echo "The factorial of $number is $result"
else
```

Docker and docker file:

Docker Structure:



- Docker client : سرویسی است که با آن می توان با هاست ارتباط برقرار کرد
  همان docker-cli می باشد که command ها را با استفاده از آن اجرا می کنیم .
- Docker daemon : درخواست های کلاینت را پذیرفته و با سیستم عامل درخصوص
  انجام فرآیندهای build و run تعامل خواهد کرد .
- Docker registry :
  ابزاری است جهت ذخیره سازی و استفاده از imageهای داکر

Docker :

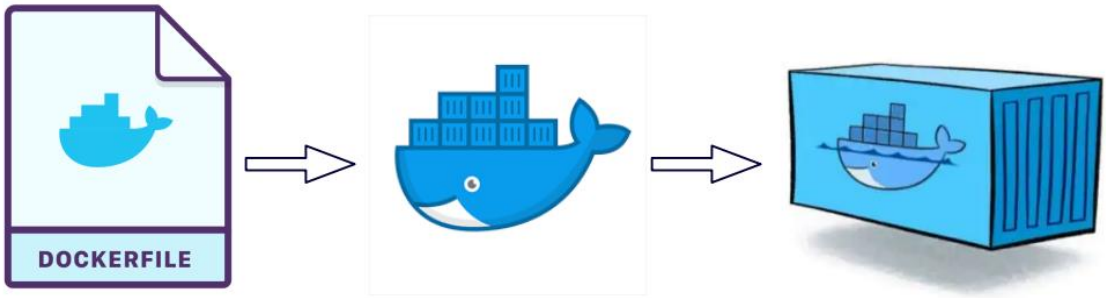
ابزاری است که به ما این امکان را می دهد که applicationهایمان را به همراه dependency هایش نصب و در یک محیط ایزوله تحت عنوان container اجرا کنیم .

Container :

برخلاف VM که دارای ساختار جداگانه برای هر host خود می باشد ، یعنی resource های مثل ram و CPU و دیسک های جداگانه و فایل های سیستمی (سیستم عامل جداگانه)دارد ، Container این امکان را به ما می دهد که با shared resource و یک سیستم عامل اما در محیط ایزوله نرم افزارهای جداگانه Up کنیم . حجم بسیار کمتری نسبت به VM خواهد داشت .

Image :

بر اساس اینکه چه appای قرار است اجرا کنیم image های مختلفی داریم . دارای پکیج ها و dependency ها است .



Docker file Docker Image Docker Container

Layer filesystem (union.fs) :

در واقع همان فایل سیستمی است که داکر برای imageهایش از آن استفاده می کند .
ساختار لایه ای دارد . به این دلیل که اگر update ای در یک لایه از image اتفاق افتاد در بروز رسانی فقط همان لایه بروز گردد و سایر لایه ها نیاز به update نداشته باشد .
در هر لایه یک Action ای دیده می شود مثلا در یک لایه یه فایل به سیستم اضافه می کند یا در لایه دیگر یک پکیج نصب می کند و ... ، تغییرات در لایه های مختلف به صورت hash شده ذخیره می گردد .

Docker benefits:

- Portability
- Isolation
- Resource efficiency

.dockerignore file:

شامل چیزهایی است که نباید در container کپی شود .
فرض کنید که در داکر فایل قصد کپی کردن تمام محتویات یک دایرکتوری را داریم اما در آن دایرکتوری یک سری فایل ها وجود دارد که لزومی به کپی ندارد (لاگ ها و ...) این فایل ها را در .dockerignore ذکر می کنیم تا کپی نشود .

Dockerfile debugging:

```
FROM ubuntu:latest  <----- داشته باشیم latest نباید تگ
RUN apt-get update
RUN apt-get install golang mariadb-server
RUN mkdir /app
WORKDIR /app/go
ADD go_project/ /app/go
RUN go mod download
RUN go build -o main .
EXPOSE 80
RUN ./main
```

- وجود تگ latest باعث می شود که اگر در تاریخی دیگر داکر فایل را اجرا کنیم و latest آن image بروز شده باشد ما یک image دیگر دانلود کنیم و شاید dependency ها تغییر کند .
- در فایل golang و mariadb نصب شده اما از image مربوط به ubuntu استفاده شده !!!!!!
- فایل دستور CMD و ENTRY POINT ندارد .

- دستورات RUN مربوط به apt-install را می توانستیم در یک خط اجرا کنیم زیرا باعث می گردد تعداد لایه ها کمتر شود .
- در این use case از app ای با زبان go و یک دیتابیس mariadb استفاده شده که باید در container های جداگانه اجرا گردند . (multi stage)
- بهتر است به جای COPY از ADD استفاده شود . دستور ADD به صورت اتوماتیک فایل های فشرده را extract و محتوی آن را کپی می کند . به همین دلیل بهتر است به جای COPY ار دستور ADD استفاده شود .
همچنین دستور ADD می تواند از URL نیز کپی انجام دهد (دانلود کند و در مسیر مربوطه قرار دهد).
- در واقع EXPOSE ۸۰ فقط جهت یادآوری اینکه app روی ۸۰ میاد بالا نوشته شده و هیچ کاری نمیکنه .

My docker container can not access 172.17.0.0/24:

رنج default مربوط به ip های داکر در همین رنج ۲۴/۱۷۲,۱۷,۰,۰ قرار دارد لذا شاید منجر به ip conflict شود لذا یک راه این است که رنج default داکر را عوض کنیم .
یک network custom با یک رنج دیگر برای container تخصیص دهیم .

Container orchestration :

- Scale container
- Load balancing
- Failover
- Network handling

دو نمونه docker swarm و kubernetes

Docker Network type:

- Bridge : default network type

در این حالت می تواند با سایر Container ها به صورت internal ارتباط برقرار کند . در واقع در این حالت container ها یک رنج ip دریافت می کنند .

- Host network

در این حالت container ای که دارای این مدل network است می تواند با host اصلی ارتباط مستقیم برقرار کند .
در این حالت نیازی به expose پورت به بیرون نیست و مستقیما در هاست دیده می شود .

- Overlay network
- Macvlan network

در این حالت یک ip مجزا به کانتینر از رنج ip هاست تخصیص داده می شود (شبیه VM)

- Ipvlan network

در این حالت یک ip مجزا به کانتینر از رنج ip هاست تخصیص داده می شود (شبیه VM)

- None network
- Custom network

Different between ENTRYPOINT and CMD :

نکته ای جداگانه در خصوص این دو مولفه این است که اگر چندین CMD در یک DockerFile وجود داشته باشد فقط آخرین CMD موثر خواهد بود .
هر دو این موارد تعریف می کنند که چه command ای وقتی Container بالا می آید باید اجرا شود .

۱. داکر فایل باید حداقل یک ENTRY POINT یا CMD داشته باشد .
۲. درواقع ENTRYPOINT دستور اجرا شونده را مشخص می کند اما CMD به ENTRYPOINT آرگومان pass می دهد .

```
FROM debian:buster
COPY . /app
RUN apt-get update
ENTRYPOINT [“bash1.sh”]
CMD [“param1”,”param2”]
```

حالا اگر در ترمینال دستور ذیل را بزنیم به شکل توضیح داده شده اجرا می گردد :

Docker run myimage —--> docker will build a container and immediately run this command in container

bash1.sh param1 param2

۳. می توان CMD را override کرد . هنگام up کردن container وقتی یک دستور را وارد می کنیم در واقع CMD را override می کنیم .

برای مثال اگر داکر فایل ما به شکل ذیل باشد :

```
FROM debian:buster
COPY . /app
RUN apt-get update
CMD [“bash1.sh”]
```

حال اگر در ترمینال دستورهای ذیل را بزنیم تفاوت محسوسی ایجاد می شود :

Docker run myimage —---> docker will build a container and immediately run bash1.sh

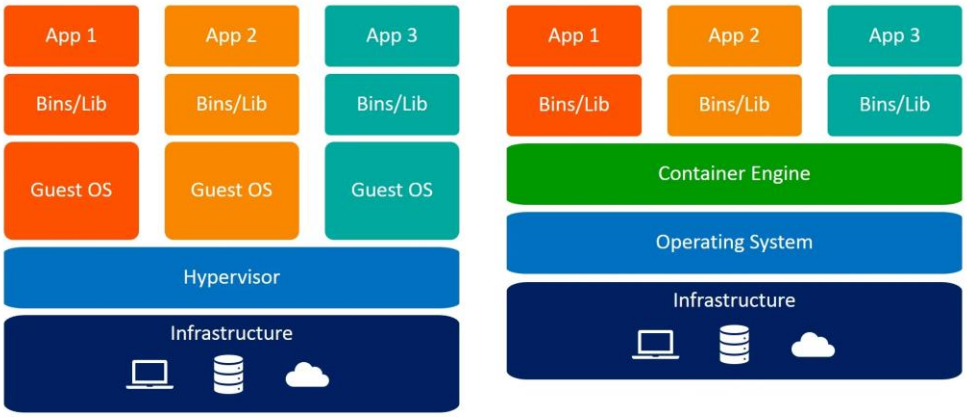
Docker run myimage bash2.sh —--> docker will build a container and run bash2.sh

به این قابلیت override شدن CMD می گویند .

Github dockerfile:

https://github.com/ehsanDadashi/DockerFile.git

Different between docker and VM :



Virtual Machines Containers

- سیستم عامل در VM مجزا است اما در docker سیستم عامل یکی است.
- فایل های سیستمی مجزا در VM وجود دارد اما در docker فایل ها مشترک است.
- کانتینر حجم بسیار کمتری نسبت به VM داره.

How to secure docker containers :

۱. استفاده از image های minimal و عدم وجود پکیج های اضافه
۲. بروز رسانی پکیج های امنیتی بر روی container ها
۳. اجرا container ها با کاربر غیر از root
۴. انجام تنظیمات firewall ای جهت محدود کردن دسترسی
۵. استفاده از Docker Content Trust جهت چک کردن signature مربوط به image
۶. مانیتور کردن activity و لاگ های audit مربوط به container ها

Docker health check:

یک script برای چک کردن وضعیت سرویس می باشد که می تواند یک http request ساده باشد .
یا یک query روی دیتابیس و ...

Cloud native apps:

- Scalable (stateless app)
- Portability
- Reliable
- Easy to merge

Docker compose:

دلیل استفاده از docker-compose :

- در سیستم های devops تلاش بر این است که تمام سرویس ها به صورت کد up شوند . (infrastructure as code)
- می توان چندین سرویس را در یک فایل مدیریت کرد

تفاوت expose و port در فایل docker compose :

- در این فایل expose اصلا پورت را برای بیرون host ، پابلیش نخواهد کرد و در واقع به خود service متصل خواهد شد . (مثلا پورت دیتابیس به این صورت expose می گردد تا از بیرون به آن دسترسی نباشد اما سایر container ها به آن دسترسی داشته باشند.)
- اما ports قابلیت دسترسی به بیرون را هم می دهد .

استفاده از دستور depends_on :

- فرض کنید یک app داریم که قرار است با دیتابیس mysql کار کند و تا وقتی که دیتابیس بالا نباشد اجرا نخواهد شد .در Compose برای app با command اشاره شده (depends_on) مشخص می کنیم که ابتدا mysql اجرا گردد و سپس app مربوطه up شود .

استفاده از restart policy :

- در صورتی که به هر دلیل سیستم crash و مجددا روشن شود باید restart تنظیم شده باشد که container اجرا شود .
- چند حالت دارد .
  1. Always ---> همیشه وقتی کانتینر پایین است ری استارت کن
  2. Unless-stopped ---> شبیه بالایی است غیر از زمانی که دستی Stop شده باشد .
  3. On-failure ---> زمانی ری استارت می گردد که با یک خطا از کانتینر خارج شده باشد .

به صورت default از فرمت yaml برای فایل های docker compose استفاده می گردد اما می توان از فرمت json نیز استفاده کرد .

Dry Run :

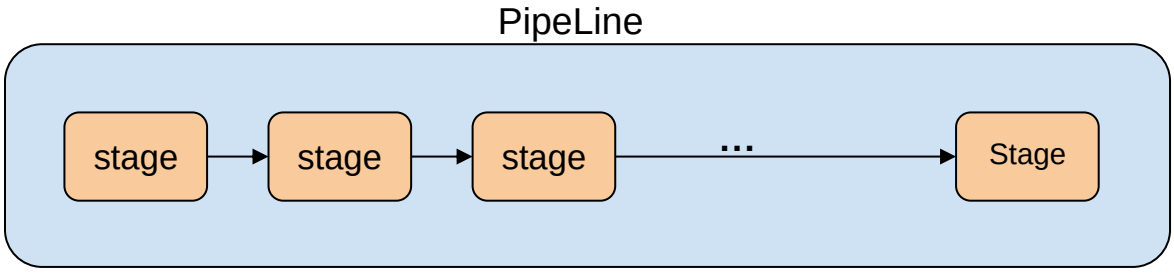
در واقع تست تغییرات قبل از عملیاتی سازی آن ها است مثال :

- در nginx با سوئیچ -t این کار انجام می شود .
- در bash script با سوئیچ -n
- در ansible با سوئیچ -- Check
- در kuber با سوئیچ -- dry run

CI/CD = Continuous Integration/Continuous Deployment

- در واقع CI مرحله تولید تا build
- در واقع CD مرحله deployment می باشد .

برای انجام CI/CD می توان از ابزارهای مختلفی استفاده کرد مثل gitlab و jenkins



حداقل دو stage باید برای pipeline وجود داشته باشد (build & deploy)

Jenkins :

دو راه برای ارتباط و ساخت CI/CD در jenkins وجود دارد

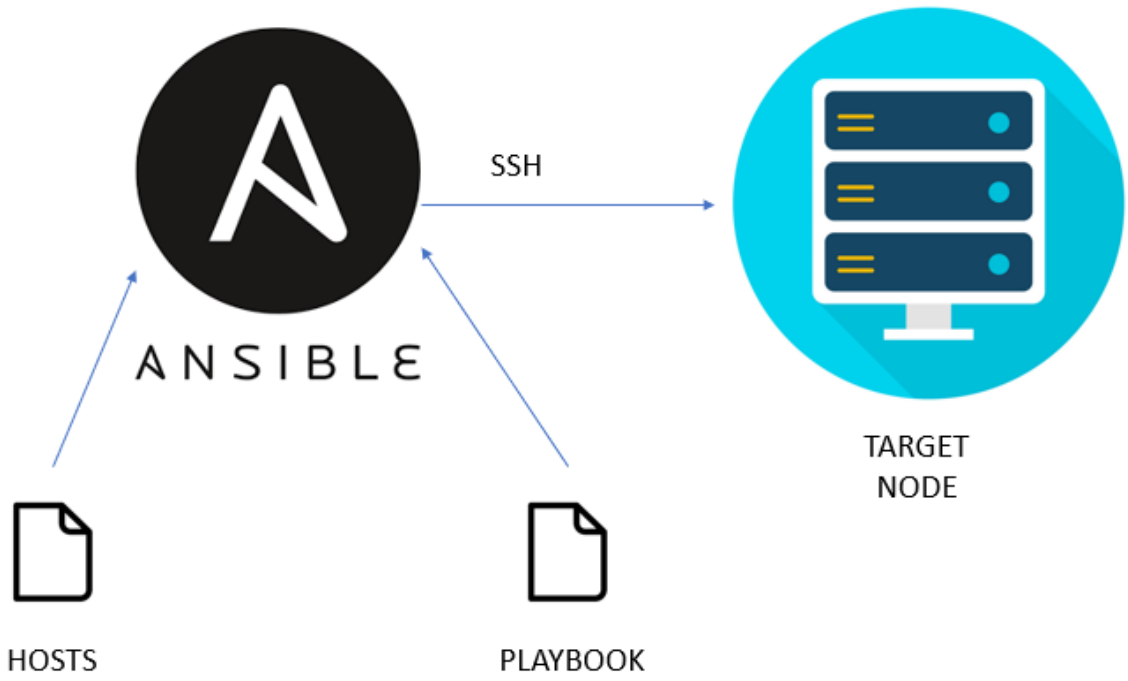
1. Jenkins UI
2. Jenkins file (groovy) —-> pipeline as a code

```
pipeline {
        agent any
        stages {
                stage (“build”) {
                        steps {
                                echo “building the app …”
                        }
                }
                stage (“test”) {
                        steps {
                                echo “testing the app …”
                        }
                }
                stage (“deploy”) {
                        steps {
                                echo “deploying the app …”
                        }
```

```
            }
        }
}
```

ANSIBLE

اجرا نکردن بخشی از playbook :
در playbook از تگ استفاده کند و هنگام اجرا
ansible-playbook myplaybook.yml --skip-tags "webserver"
اجرای playbook روی ۵۰ سرور به صورت همزمان :
در کانفیگ ansible یعنی فایل ansible.cfg پارامتر forks را برابر ۵۰ قرار می دهیم .



IPS/IDS vs Firewall

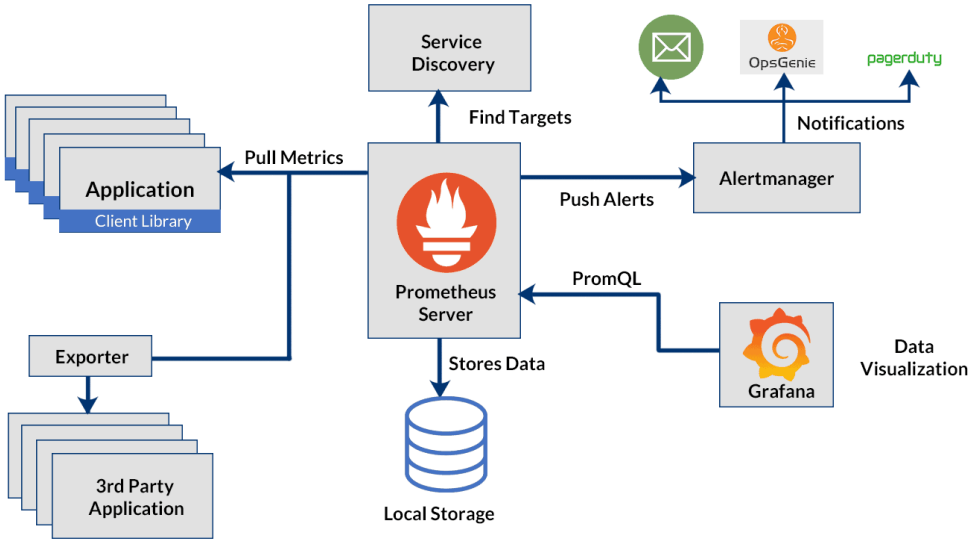
- Firewall ---> port check
  Ip check (access list , deny list)
- IPS/IDS ---> check viruses & malware & …
  Check signature

  IDS ---> detection (خواهد کرد اما خبر دار اجازه ورود می دهد)
  IPS ---> protection (جلوگیری از ورود)
- SIEM ---> ارسال لاگ از تمام سرور ها به سرور مرکزی و تصمیم گیری در خصوص آن
  چک کردن فایل های حساس و مسیر های حساس

لیست تمام processها
لیست پورت های باز و ...
تشبیه به این صورت است که firewall مانند یک نگهبان در ورودی است که یک لیست دارد و در صورتی که نام فرد در آن لیست موجود باشد اجازه ورود به وی را می دهد .
اما چک نمی کند آیا علی وسیله ای غیرمجاز به همراه دارد یا خیر؟

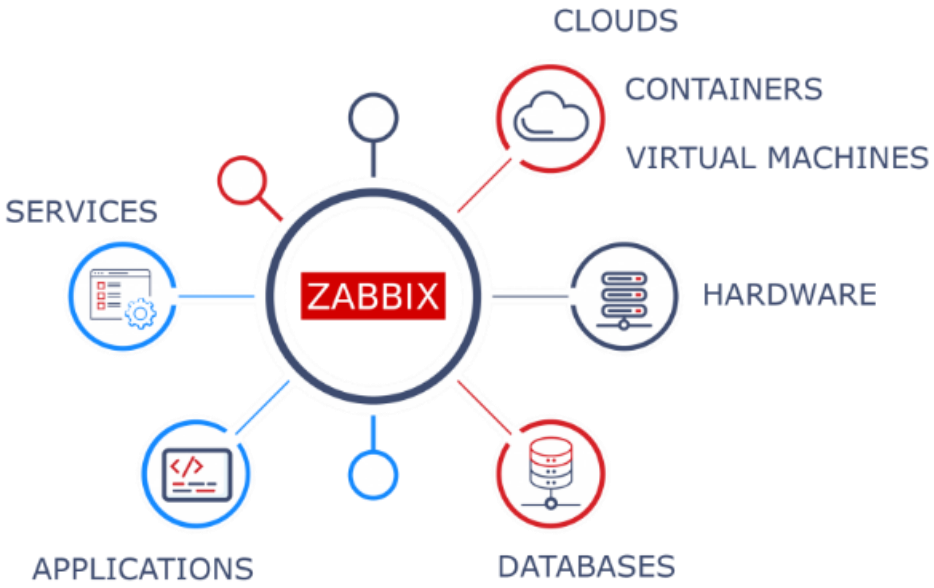
Monitoring

Prometheus



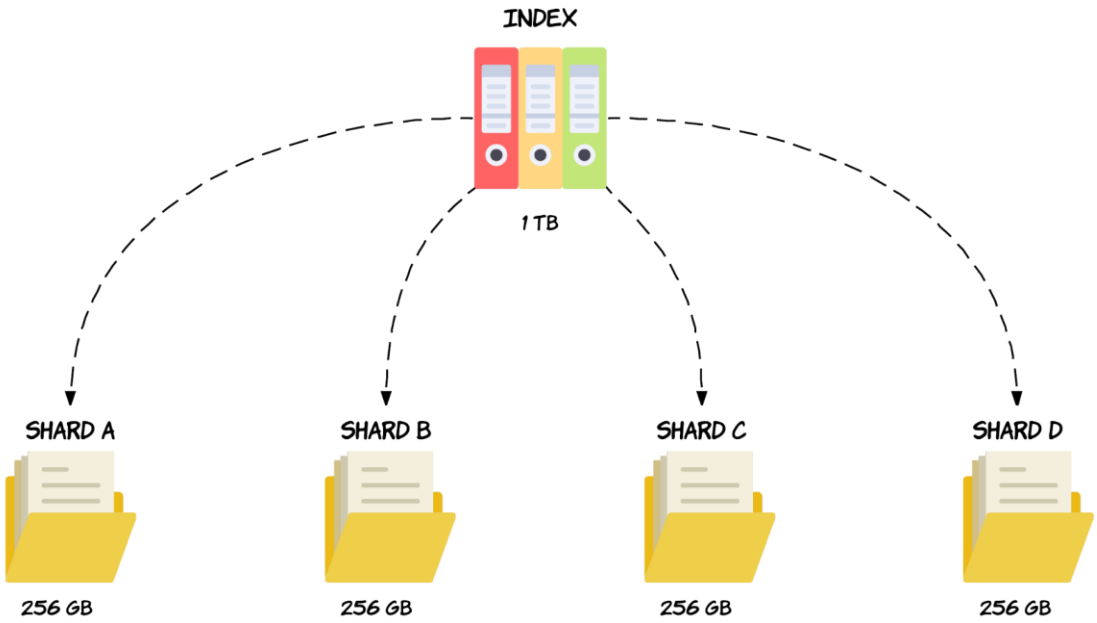
- تمام داده ها به صورت Key:value می باشد . (مثل redis)
- داده ها numeric می باشد و string نداریم .
- برای هر سرویس exporter مختص خود را داریم .

zabbix

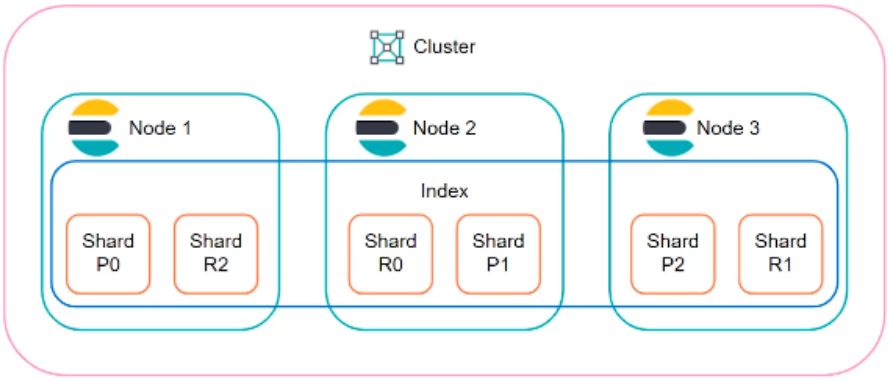


replica vs shard

- Replica : دقیقا یک کپی از داده قبلی است
- Shard : به فرض اگر داده کلا ۱ ترابایت باشد و ۴ شرد داشته باشیم هر شرد ۲۵۶ گیگ از دیتا را خواهد داشت.



ELK CLUSTER

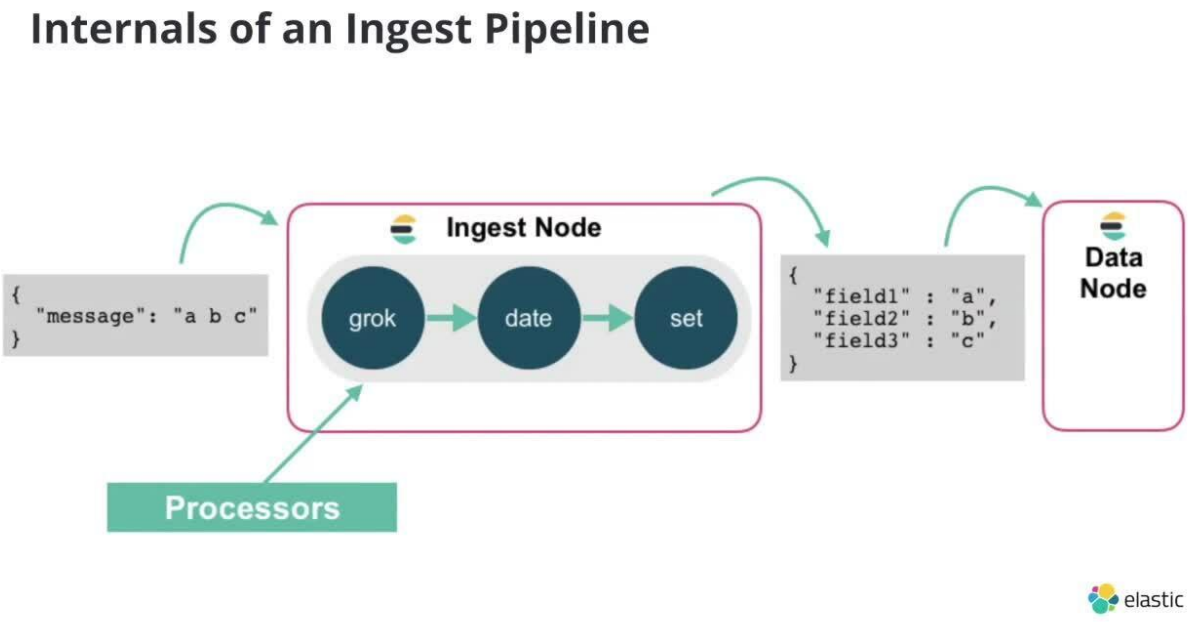


چرا باید تعداد نودهای cluster فرد باشد ؟

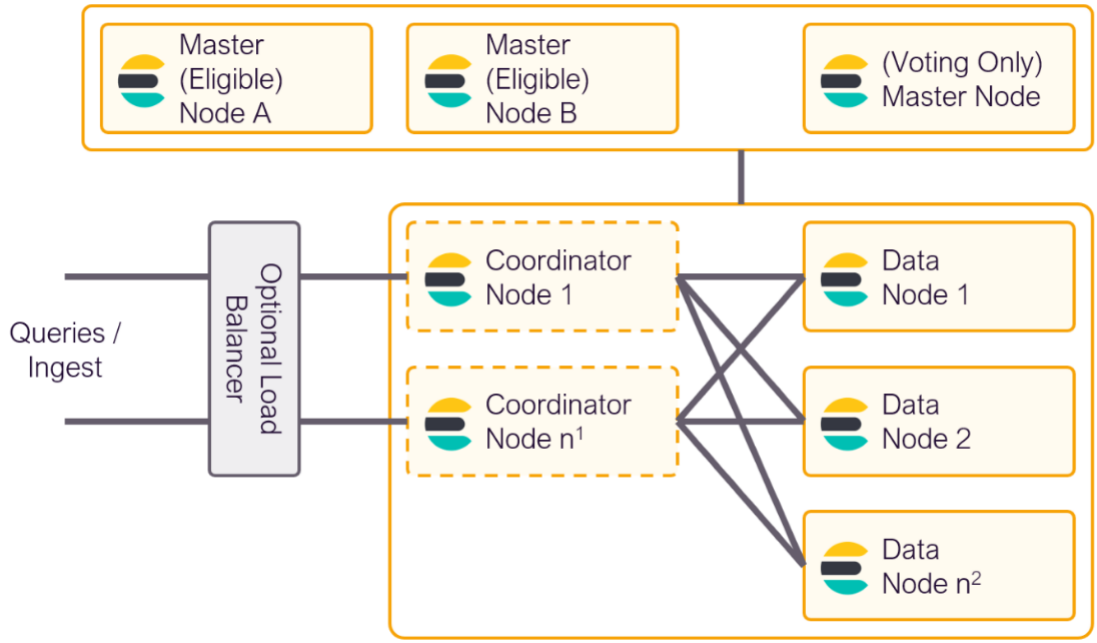
- چون در رای گیری برای انتخاب master یک نود رای بیشتری بیاورد . (witness node)

ELK cluster node rolls:

- Master : cluster management
  وظایفی مانند ایجاد/حذف ایندکس ها مدیریت shard allocation و حفظ وضعیت کلی کلاستر
- Data :
  ذخیره و مدیریت index ها و انجام عملیات search و analyze
- Ingest : process data before indexing (change format , parsing , …)



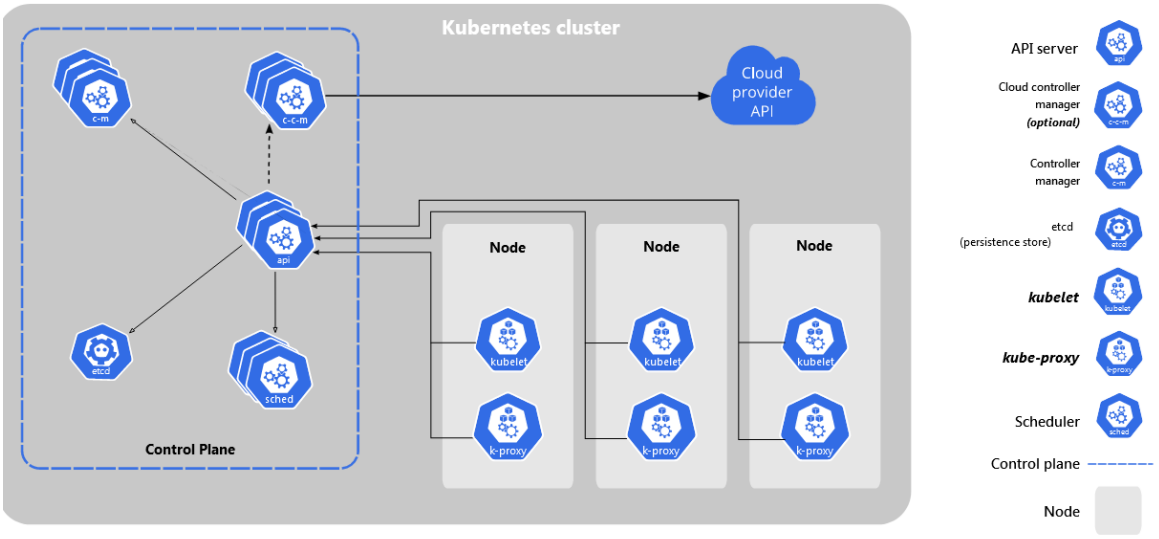
- Coordinate : it is a data less node that receive data and send it to other node
- Machine learning : it manage and runs machine learning process
- Voting only : it is a node that just vote about choose master node and it is not be able to be master



- Remote cluster : it is a client that will be used to connect to elasticsearch cluster. It will be use to cross cluster search

Kubernetes

- Pod کوچکترین مولفه کوبر که می تواند شامل یک یا چندین کانتینر باشد .
- API server هندل کردن درخواست های روی کوبر
- ETCD پایگاه داده کوبر
- Sched قانون گذار و تخصیص منابع به APP
- C.M مانیتور کردن وضعیت کلاستر - افزودن feature
- Kublet kuber agent
  وظیفه ارتباط با api مربوط به master و همچنین ارتباط با container runtime interface در خود worker را دارد .
- Kube proxy وظیفه ارتباط شبکه را به عهده دارد



نحوه ارتباط با kuber

در واقع با این ابزار ها می توان با API Server ارتباط برقرار کرد .

- Interactive : kubectl
- Non-interactive : manifest ---> yaml file

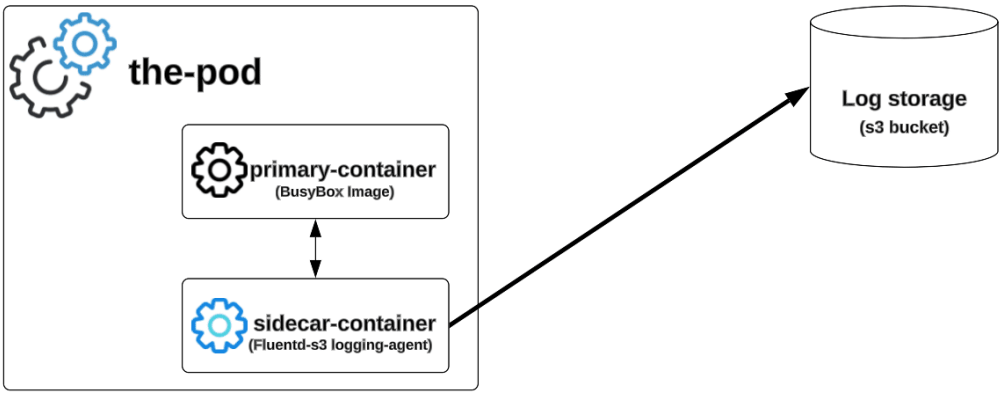
namespace

ایزوله کردن چند اکوسیستم از یکدیگر .

مثلا برای ایجاد محیط های متفاوت از یک APP ، محیط test و prelive و production و ...

Sidecar

یک کانتینر در کنار کانتینر اصلی است که یک وظیفه جانبی مثل ارسال لاگ و ... را بر عهده دارد .



Kubeadm ---> ابزار نصب kuber

Statefulset --->

اجرای app های stateful در kuber

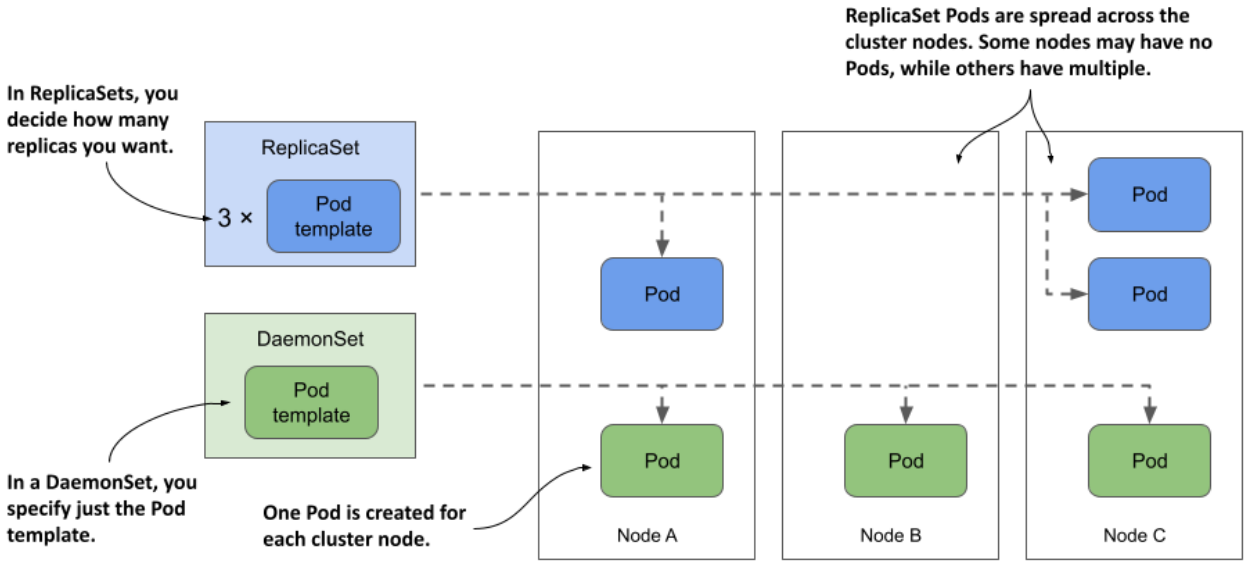
Deployment --->

اجرای app های stateless در kuber در این روش به تعدادی که شما در تعریف deployment اعلام می کنید (replica) در پیاده سازی POD خواهید داشت.

Daemonset ---> ها باید وجود داشته باشد worker روی تمام

در واقع روی تمام worker ها این pod وجود خواهد داشت .

مثلا یک fluentd که لاگ تمام pod ها را برای Elastic بفرستد .



Cronjob ---->

جهت run کردن job های cron ای در سطح kubernetes

Service ---->

جهت ارتباط با POD ها و گرفتن سرویس از آن POD ها استفاده می شود .

Service type :

- Cluster ip ---- در سطح کلاستر کوبر سرویس در دسترس است و به بیرون دسترسی نمی دهد
- Node port ---- یک پورت مثل ۳۰۰۵۰ به سرویس تخصیص می دهد و از بیرون قابل دسترسی است.در تمامی ورکرها در دسترس است .

با استفاده از سرویس به صورت خودکار load balance انجام می گردد.

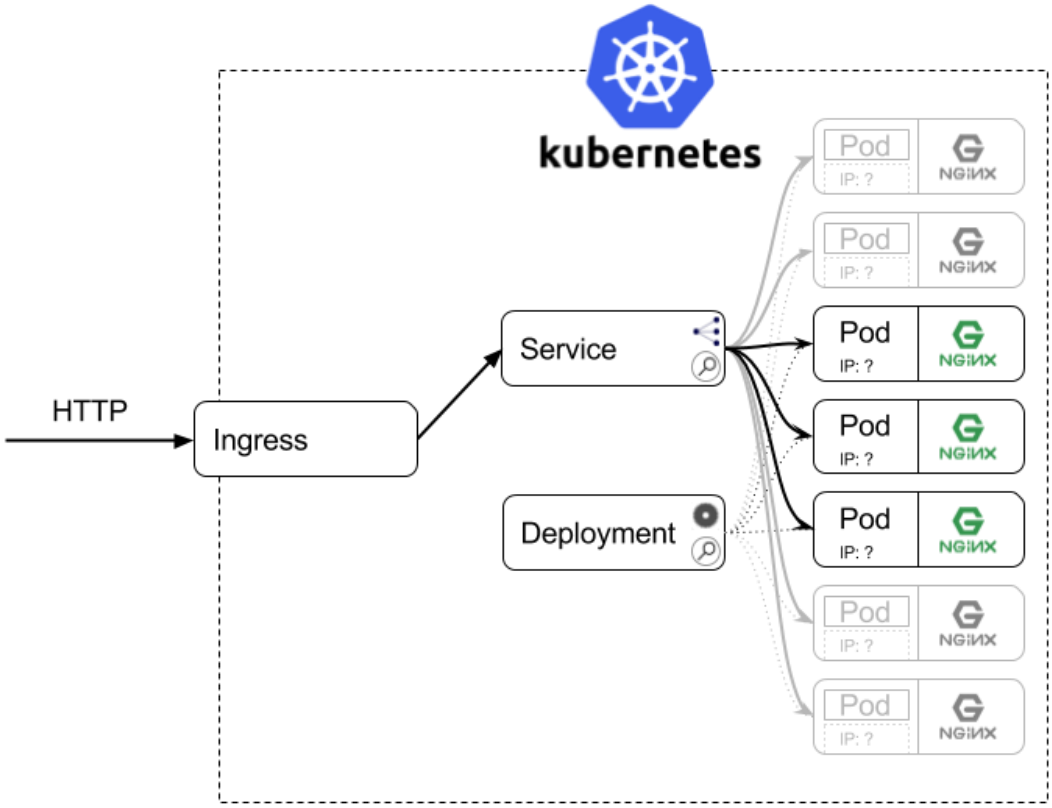
Ingress ---- nginx , haproxy

همان کار وب سرور را می کند .

مثال :

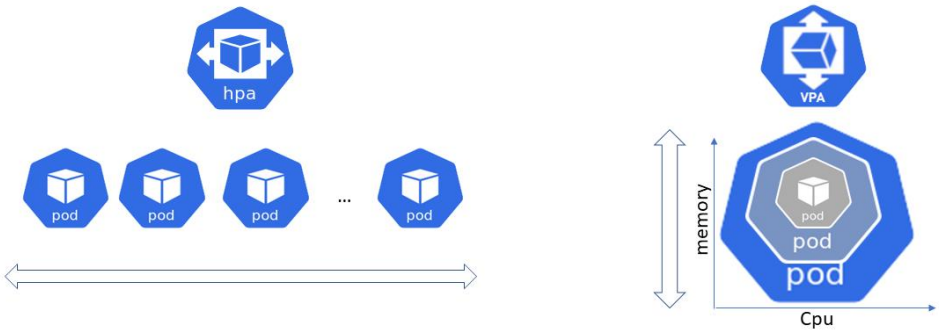
If / --> service test --> pod test app

If /login --> service login --> pod login app

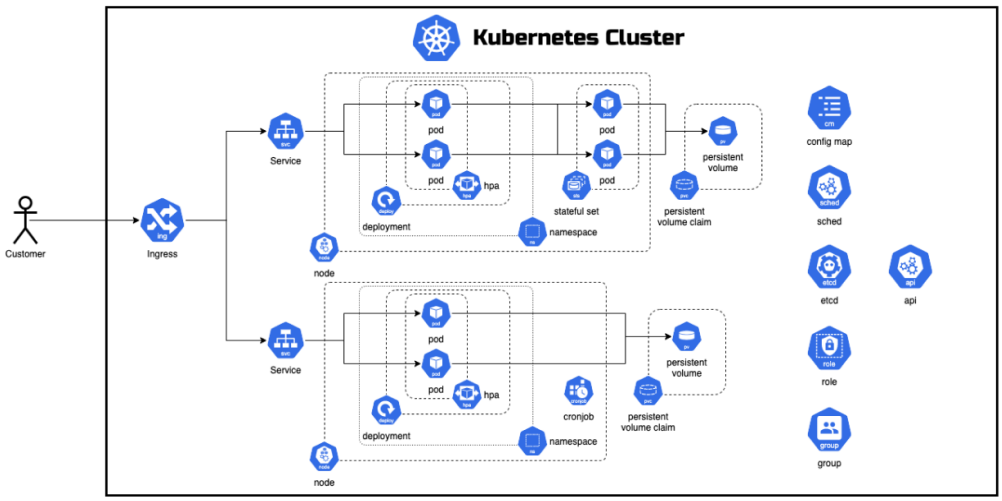


HPA : Horizontal Pod Autoscaler

در واقع با یک شرط (مثلا میزان مصرف رم و ...) auto scale را به صورت افقی برای POD ها انجام می دهد .



در نهایت شکل کامل کلاستر kubernetes به شکل ذیل خواهد بود



Kubernetes manifest file :

Manifest deployment:

apiVersion: apps/v1 ---> به ورژن کوبر اشاره می کند

Kind: Deployment ---> re

source Type

(Deployment | statefulset | daemonset | service , ...)

metadata : ---> اطلاعات مربوط به خود کوبر

name: nginx-deployment نام اختصاص داده شده به منبع در کوبر

labels: جهت دسته بندی منابع و برچسب زدن به آن استفاده می گردد

برای مثال به POD و SERVICE و DEPLOYMENT ای که مربوط به nginx می باشد label ای مشابه (برای مثال nginx) نسبت می دهیم .

app: nginx

spec: ---> اطلاعات مربوط به اپلیکیشن

replicas: 3

selector:

matchlabels: ---> اشاره به برچسبی که در بالا تعریف شده است .

app: nginx

template: ---> نگهداری می گردد POD در این بخش اطلاعات مربوط به

metadata:

labels:

app: nginx

spec: ---> محل تعریف POD

containers:

- name: nginx

Image: nginx:1.14.2

ports:

- containerPort: 80

Manifest service:

apiVersion: v1

```
kind: Service
metadata:
        name : nginx
spec:
        selector:
                app: nginx
        ports:
                - protocol : TCP
                port : 8080
                targetPort: 80
```

Github kubernetes with nginx in cluster:
https://github.com/ehsanDadashi/k3s.git

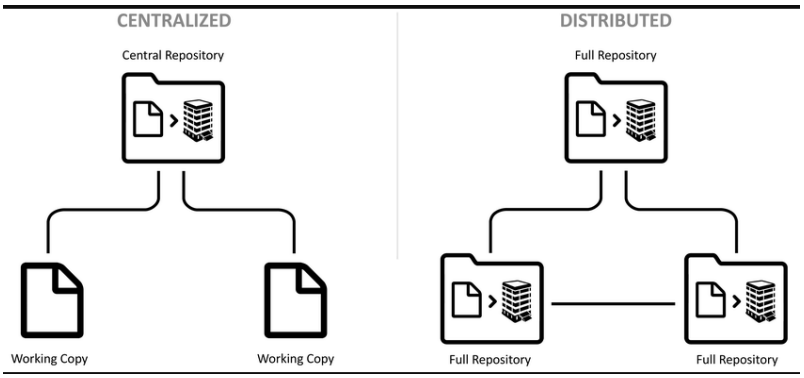
Version control

git:

مزایای یک source control :

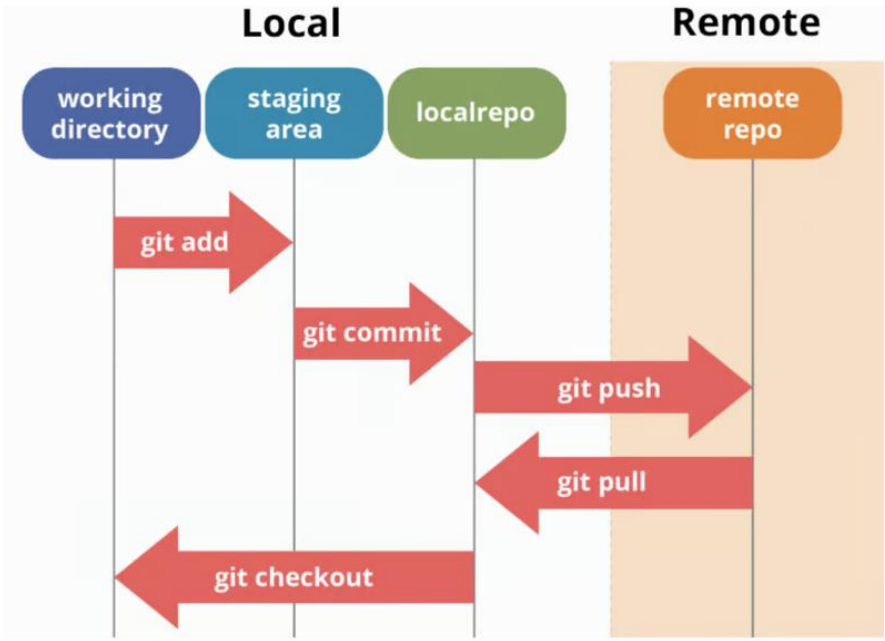
۱. امکان track کردن history کد وجود خواهد داشت .
۲. چند نفر همزمان روی یک کد کار خواهند کرد .

Difference between centralized and distributed control system :



۱. در سیستم centralized :
   - هیچ developer ای تمام کد را در اختیار ندارد .
   - یک سرور مرکزی برای ذخیره تمام نسخ وجود دارد .
   - اگر سرور مرکزی crash کند تمام داده ها از بین خواهد رفت .
۲. در سیستم distributed:
   - کد در سیستم تمام developer ها وجود دارد .
   - در سرور مرکزی هم ذخیره می گردد.
   - اگر سرور مرکزی crash کند تمام داده ها از بین خواهد رفت .

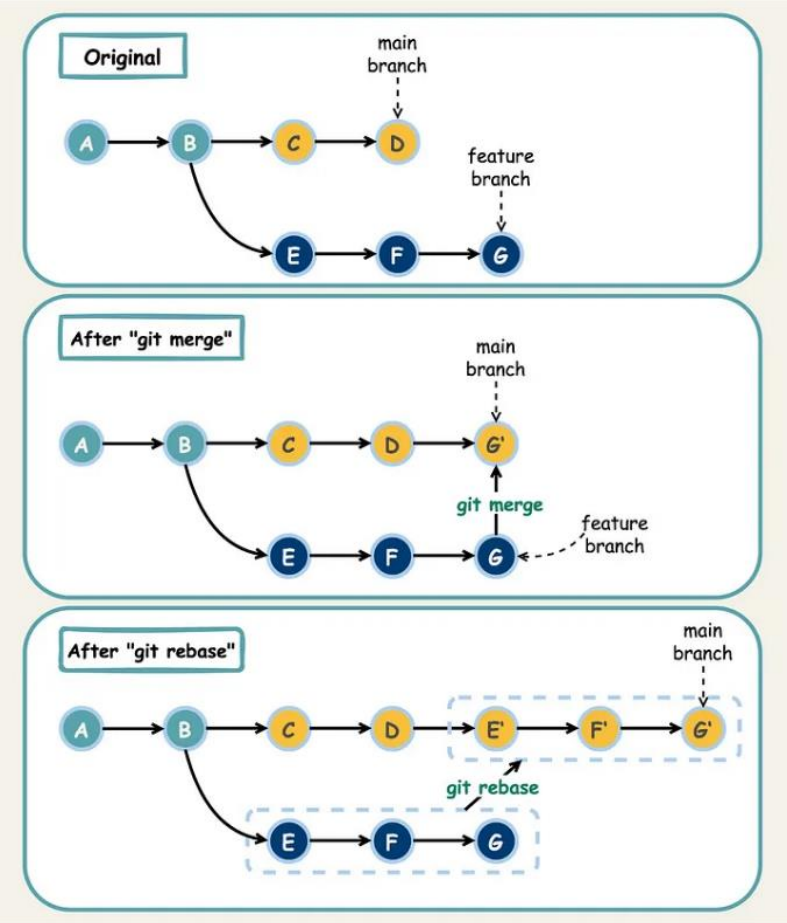
git workflow:



Git commands :

- git init —--> ایجاد یک repository جدید
- git add —--> اضافه کردن فایل های جدید به stage area
- git commit —--> ارسال تغییرات ایجاد شده به repository
  در واقع با هر commit بخش stage خالی نخواهد شد .
- git status —--> نمایش وضعیت جاری working dir و staging area
- git checkout —--> کار کردن با نسخ مختلف یک کد (جا به جایی بین commit ها)
  از این دستور برای جا به جایی بین branch هم امکان پذیر است
  git checkout <branchname>
- git clone —--> download all repo from git to your computer
- git diff —--> show change between working dir and stage area
- git push —--> push your file to remote dir
- git remote set-url origin https://github.com/ehsanDadashi/techno-message-parser.git
  git pull origin main
  git push origin main —--> ارسال اطلاعات local به github

Difference between rebase and merge:



- git merge :

  این دستور تاریخچه هر دو branch را حفظ می کند .
  عیب این روش این است که در صورتی که تعداد ادغام ها زیاد باشد تاریخچه پیچیده می گردد .

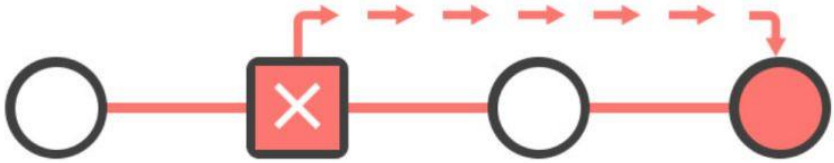
- git rebase:

  این دستور تاریخچه git را تحت تاثیر قرار می دهد و تاریخچه را بازنویسی می کند .

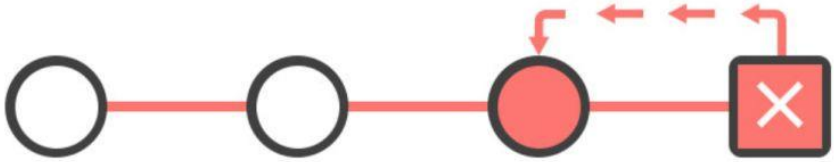
Difference between revert and reset:

هر دو مورد به درد زمانی می خورد که اشتباهی رخ داده است و می خواهیم به وضعیت سابق بازگردیم.

Reverting



Resetting



در reset هم به نسخه قبلی rollback می کنیم و هم تاریخچه دستور به عقب باز می گردد (در واقع pointer از روی نسخه جاری به نسخه قبلی اشاره می کند)لذا تاریخچه داخل سیستم با تاریخچه سرور متفاوت خواهد بود (Conflict دارد) .

اما در revert همان عمل انجام می شود ولی به جای اینکه pointer مربوطه به عقب برگردد ، یک کپی از نسخه قبل به عنوان نسخه بعدی در نظر می گیرد لذا در این حالت انگار نسخه جدیدی ارائه شده که با نسخه قبل از نسخه جاری برابر است و در این حالت ما conflict مربوط به history را در سیستم و سرور نخواهیم داشت .