* دستور کل کانتینر های stop, start و dead را نشان می دهد. #docker ps -qa
* دستور کل کانتینر ها را حذف می کند. docker rm -v -f $(docker ps -aq)#
* دستور نمایش ورژن سیستم عامل #cat /etc/os-release
* **روش ساخت image سفارشی در داکر**

اگر نیاز به image سفارشی دارید، یک Dockerfile ایجاد کنید:

**FROM** node:14

**WORKDIR** /app

**COPY** package.json .

**RUN** npm install

**COPY** . .

**CMD** ["npm", "start"]

در فایل‌های Dockerfile،  **ARG**  یک دستور است که برای تعریف متغیرهای قابل عبور در زمان ساخت (build-time variables) استفاده می‌شود. این متغیرها فقط در مرحله ساخت Docker image قابل استفاده هستند و در زمان اجرای کانتینر در دسترس نیستند.

کاربرد ARG در Dockerfile:

* این دستور به شما امکان می‌دهد مقادیری را در زمان ساخت (docker build) از طریق خط فرمان یا فایل‌های دیگر پاس دهید.
* معمولاً برای پارامترهایی مثل نسخه‌های نرم‌افزار، مسیرهای نصب، یا تنظیمات وابسته به محیط استفاده می‌شود.
* **Docker Compose**

**ساخت و اجرای** **Docker Compose**

#**apt install docker-compose**دستور نصب داکر کامپوز

#**apt purge docker-compose** دستور حذف داکر کامپوز

#**docker –version** دستور نمایش ورژن داکر

#**docker-compose –version** دستور نمایش ورژن داکر کامپوز

#**mkdir** my-docker-project

#**cd**  my-docker-project

#**nano docker-compose.yml**

#**docker-compose up -d**

#**docker-compose ps**

#**docker-compose logs**

#**docker-compose down**

#**docker-compose down -v** دستور حذف داکر کامپوز با ولوم هاش

#**docker-compose pull**

* مثال:

*root@devops:/home/debian/wordpress#* **ls**

db **docker-compose.yml** **docker-compose.yml.bk** wordpress

* **Example1 docker-compose:** 
  + **WordPress**
  + **MariaDB**

فایل داکر کامپوز با هر دو پسوند yml , yaml میتونه نوشته بشه.

**File: docker-compose.yml**

**version: "3.9"**

**services:**

**wordpress:**

**image:** docker.arvancloud.ir/wordpress

**container\_name:** wordpress

**restart:** unless-stopped

**ports:**

- "8080:80"

**environment:**

WORDPRESS\_DB\_HOST: db:3306

WORDPRESS\_DB\_USER: wordpressuser

WORDPRESS\_DB\_PASSWORD: wordpress\_password

WORDPRESS\_DB\_NAME: wordpress

**volumes:**

- ./wordpress:/var/www/html

- /etc/localtime:/etc/localtime:ro

**depends\_on:**

- db

**networks:**

- wordpress\_network

**db:**

**image:** docker.arvancloud.ir/mariadb

**container\_name:** wordpress\_db

**restart:** unless-stopped

**environment:**

MYSQL\_ROOT\_PASSWORD: your\_root\_password

MYSQL\_DATABASE: wordpress

MYSQL\_USER: wordpress

MYSQL\_PASSWORD: wordpress\_password

**volumes:**

- ./db:/var/lib/mysql

- /etc/localtime:/etc/localtime:ro

**networks:**

- wordpress\_network

**networks:**

wordpress\_network:

**driver:** bridge

- **targets:** ['node-exporter:9100']

**troubleshooting**

#**docker logs -f** mariadb

صحت سنجی فایل کانفیگ داکر کامپوز #**docker-compose config**

#**rm -rf** wordpress/ db/ docker-compose.yml.bk حذف فایل و فولدر های مشخص شده مسیر جاری

* **Example2 docker-compose:**
* **Prometheus** (metrics *collection* and storage)
* **Node Exporter** (host system *metrics*)
* **Grafana** (*visualization*)
* Configuration Files:
  + Docker-config.yml
  + Prometheus.yml

*File: docker-config.yml*

**version:** '3.9'

**services:**

**node-exporter:**

**image:** docker.arvancloud.ir/prom/node-exporter:latest

**container\_name:** node-exporter

**restart:** unless-stopped

**volumes**:

- /proc:/host/proc:ro

- /sys:/host/sys:ro

- /:/rootfs:ro

**command:**

- '--path.procfs=/host/proc'

- '--path.sysfs=/host/sys'

- '--collector.filesystem.ignored-mount-points=^/(sys|proc|dev|host|etc)($$|/)'

**ports:**

- '9100:9100'

**networks:**

- monitoring

**prometheus:**

**image:** docker.arvancloud.ir/prom/prometheus:latest

**container\_name:** prometheus

**restart:** unless-stopped

**volumes:**

- ./prometheus.yml:/etc/prometheus/prometheus.yml

**command:**

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

**ports**:

- '9090:9090'

**depends\_on:**

- node-exporter

**networks:**

- monitoring

**grafana:**

**image**: docker.arvancloud.ir/grafana/grafana:latest

**container\_name**: grafana

**restart**: unless-stopped

**volumes**:

- grafana-storage:/var/lib/grafana

**environment:**

- GF\_SECURITY\_ADMIN\_USER=admin

- GF\_SECURITY\_ADMIN\_PASSWORD=admin

**ports:**

- '3000:3000'

**depends\_on:**

- prometheus

**networks:**

- monitoring

**networks:**

monitoring:

**driver:** bridge

**volumes:**

grafana-storage:

**driver:** local

*prometheus.yml File:*

**global:**

**scrape\_interval:** 15s

**scrape\_configs**: تعریف منابع متریک

- **job\_name:** 'prometheus'

**static\_configs:**

- **targets:** ['localhost:9090']

- **job\_name:** 'node-exporter'

**static\_configs:**

**-targets:** ['node-exporter:9100']

* **Restart Policies (سیاست‌های راه‌اندازی مجدد) در Docker**

وقتی یک کانتینر به هر دلیلی متوقف می‌شود، Docker می‌تواند به صورت خودکار آن را مجددا راه‌اندازی کند. این رفتار با استفاده از سیاست‌های راه‌اندازی مجدد (Restart Policies) کنترل می‌شود.

1. **no** (پیش‌فرض) : کانتینر به‌هیچ‌ وجه به‌صورت خودکار راه‌اندازی نمی‌شود، حتی اگر به‌صورت دستی Docker را ریستارت کنید.
2. **always:** کانتینر همیشه پس از توقف راه‌اندازی مجدد می‌شود، حتی اگر شما آن را به‌صورت دستی متوقف کرده باشید docker stop اگر Docker daemon ریستارت شود، container دوباره شروع می‌شود. معمولا برای کانتینر هایی که مهمند استفاده می شوند که همیشه باید بالا باشند. مثل وب سرور..
3. **:on-failure**کانتینر فقط در صورت خروج با کد خطا (Exit Code ≠ 0) راه‌اندازی مجدد می‌شود. می‌توانید حداکثر تعداد راه‌اندازی مجدد را مشخص کنید
4. **: unless-stopped**محبوب‌ترین گزینه برای سرویس‌های پایدار. کانتینر به‌طور خودکار راه‌اندازی می‌شود، مگر اینکه به‌صورت دستی (توسط خود ادمین) متوقف شده باشد (docker stop).برخلاف always، اگر شما کانتینر را دستی متوقف کنید، حتی پس از ریستارت Docker daemon، کانتینر دوباره شروع نمی‌شود.

restart: unless-stopped

* rmdir دستور حذف دایرکتوری خالی
* rm -r دستور حذف دایرکتوری پر
* rm -rf فایل‌ها و دایرکتوری‌ها را حذف می‌کند
* دستور **docker** **kill** برای متوقف کردن فوری (terminate کردن) یک یا چند container در حال اجرا در docker استفاده می‌شود. این دستور برخلاف docker stop که به کانتینر فرصت می‌دهد تا به صورت graceful خاتمه یابد، کانتینر را بلافاصله و بدون مراحل عادی خاموش شدن می‌بندد.
* **انواع مانیتورینگ Pull و Push**

**:Pull Monitoring (**مثال Prometheus)

**ویژگی‌ها:** سرور مانیتورینگ داده‌ها را از Targets می‌کشد، Targets را از طریق یک Endpoint در دسترس قرار می‌دهند، سرور تصمیم می‌گیرد که چه زمانی و چگونه داده‌ها را جمع‌آوری کند.

**مزایا:** کنترل متمرکز بر روی فرکانس جمع‌آوری داده‌ها، کشف خودکار سرویس‌های جدید (Service Discovery)، امنیت بهتر (فایروال فقط نیاز به اجازه خروجی دارد)، مناسب برای محیط‌های دینامیک مثل کانتینرها

معایب:

1. نیاز به Endpoint قابل دسترس از طرف سرور مانیتورینگ

2. تأخیر در دریافت داده‌های لحظه‌ای

**سیستم Push Monitoring**

مثال: Graphite, StatsD

ویژگی‌ها:

* اهداف (کلاینت‌ها) داده‌ها را به سمت سرور مانیتورینگ می‌فرستند
* کلاینت‌ها تعیین می‌کنند چه زمانی داده ارسال شود
* معمولاً از پروتکل‌های UDP یا TCP استفاده می‌کنند

مزایا:

1. مناسب برای سیستم‌های موقت یا زودگذر (Ephemeral)

2. انعطاف‌پذیری بیشتر در ارسال داده‌ها

3. عملکرد بهتر در مقیاس بزرگ

4. مناسب برای داده‌های رویدادمحور (Event-based)

معایب:

1. نیاز به پیکربندی هر کلاینت به صورت جداگانه

2. چالش‌های امنیتی (نیاز به باز کردن پورت‌های ورودی)

3. مشکل در کشف خودکار سرویس‌ها

**ترکیب هر دو سیستم**

در بسیاری از محیط‌های مدرن از ترکیب هر دو روش استفاده می‌شود:

* Prometheus برای جمع‌آوری متریک‌ها (Pull)
* Grafana برای نمایش داده‌ها
* Alertmanager برای ارسال اعلان‌ها (Push)

**مقایسه Docker Compose و Docker Swarm**

**Docker Compose**

ویژگی‌های اصلی:

* **ابزار توسعه محور**: عمدتاً برای محیط‌های توسعه و تست استفاده می‌شود
* **مدیریت تک‌هاست**: روی یک ماشین واحد اجرا می‌شود
* **تعریف سرویس‌ها با YAML**: فایل docker-compose.yml برای تعریف سرویس‌ها، شبکه‌ها و volumeها
* **مناسب برای محیط‌های کوچک**: اجرای چندین کانتینر به صورت هماهنگ

موارد استفاده:

* توسعه و تست اپلیکیشن‌های چندکانتینری
* محیط‌های استیجینگ ساده
* راه‌اندازی سریع stackهای محلی

محدودیت‌ها:

* عدم پشتیبانی از clustering به صورت پیش‌فرض
* قابلیت‌های محدود در مقیاس‌پذیری
* عدم وجود مکانیزم‌های خودترمیمی (self-healing)

**Docker Swarm**

ویژگی‌های اصلی:

* **اورکستراسیون کانتینر**: سیستم clustering داخلی داکر
* **مدیریت چندهاست**: پشتیبانی از چندین نود (مدیر و worker)
* **مقیاس‌پذیری**: امکان scale کردن سرویس‌ها به صورت افقی
* **تحمل خطا**: قابلیت self-healing و replication
* **تعادل بار**: load balancing داخلی

موارد استفاده:

* محیط‌های تولیدی (production)
* استقرارهای مقیاس‌پذیر
* سیستم‌های با نیاز به دسترس‌پذیری بالا

محدودیت‌ها:

* پیچیدگی بیشتر در راه‌اندازی و مدیریت
* ویژگی‌های کمتر نسبت به Kubernetes
* **DOCKER SWARM**

**Docker Swarm برای استقرار برنامه‌های توزیع‌شده در چندین سرور (خوشه یا Cluster) استفاده می‌شود. Docker Compose برای اجرای چند سرویس به صورت محلی (مثلاً یک برنامه شامل دیتابیس، بک‌اند، و فرانت‌اند) روی یک سیستم طراحی شده است.**

**نکته: در زمان استفاده از docker swarm دیگر نیازی به live-restore=true در daemon.json نیست چون کنترل کانتینر ها با داکر سوارم خواهد بود.**

)sudo systemctl restart systemd-networkd **دستور تغییر آیپی سرور (**

**دستور :**

**#docker swarm init**

**#docker node ls نمایش لیست نود های منیجر و ورکر**

**نکته: روی manager node باید پورت 2377 مجاز باشد.**

**دستور ایجاد سرویس روی دو تا ورکر**

# docker service create --name web --replicas 3 -p 80:80 docker.arvancloud.ir/nginx

docker service create --name web --replicas 3 -p 80:80 docker.arvancloud.ir/alpine ping google.com

#docker service ps web

#docker service scale web=0 حذف یک سرویس

#docker service rm …

دستور تغییر تعداد سرویس ها

# docker service scale web=2

نمایش سرویس های فعال

# docker service ps web

* **The following porte must be available:**
* TCP port **2376** for secure Docker client communication. This port is required for Docker Machine to work. Docker Machine is used to orchestrate Docker hosts.
* TCP port **2377**. This port is used for communication between the nodes of a Docker Swarm or cluster. It only needs to be opened on manager nodes.
* TCP and UDP port **7946** for communication among nodes (container network discovery).
* UDP port **4789** for overlay network traffic (container ingress networking).

**راهنمای کامل Docker Compose و مدیریت با Docker Swarm**

در این بخش، نحوه ایجاد فایل docker-compose.yml و مدیریت آن با **Docker Swarm** را توضیح می‌دهیم.

**1. ساخت فایل**docker-compose.yml

مثال برای یک برنامه تحت وب با **Nginx** و **Redis**:

version: '3.8'

services:

web:

image: nginx:alpine

ports:

- "80:80" *# پورت 80 هاست به پورت 80 کانتینر*

deploy:

replicas: 3 *# تعداد کپی‌های سرویس*

update\_config:

parallelism: 2 *# تعداد کانتینرهایی که همزمان آپدیت می‌شوند*

delay: 10s *# تأخیر بین آپدیت‌ها*

restart\_policy:

condition: on-failure *# ریستارت در صورت خطا*

networks:

- app-net *# اتصال به شبکه*

redis:

image: redis:alpine

deploy:

replicas: 1

volumes:

- redis-data:/data *# ذخیره داده‌های Redis*

networks:

- app-net

networks:

app-net:

driver: overlay *# شبکه برای ارتباط بین نودها*

volumes:

redis-data:

driver: local *# ذخیره‌سازی داده‌ها روی هاست*

**نکات مهم:**

* بخش deploy مخصوص تنظیمات **Docker Swarm** است.
* overlay network برای ارتباط بین چندین سرور استفاده می‌شود.
* volumes برای ذخیره‌سازی دائمی داده‌ها (مثل دیتابیس).

دستور تبدیل worcker به manager

پیشنهاد میشه تعداد داکر منیجر هامون 3 الی 7 تا باشه

#docker node promote srv-02

#docker node demote srv-02

**2. راه‌اندازی Docker Swarm**

برای تبدیل سرور به **Swarm Manager** اجرا کنید:

docker swarm init

docker swarm init –advertise-addr 192.168.239.128

(Add --advertise-addr <IP> if you have multiple interfaces)

**3. اجرای سرویس‌ها در Swarm**

با دستور زیر docker-compose.yml را به Swarm ارسال کنید:

**docker stack deploy -c docker-compose.yml myapp**

**.4 مدیریت سرویس‌ها**

**مشاهده سرویس‌های در حال اجرا**

docker service ls

**مشاهده جزئیات یک سرویس**

docker service inspect myapp\_web

**تغییر تعداد کپی‌های سرویس (Scale)**

docker service scale myapp\_web=5

**مشاهده لاگ‌های سرویس**

bash

Copy

Download

docker service logs myapp\_web

**آپدیت یک سرویس (مثلاً تغییر نسخه ایمیج)**

bash

Copy

Download

docker service update --image nginx:latest myapp\_web

**5. دستورات مهم مدیریت Swarm**

| **دستور** | **توضیحات** |
| --- | --- |
| docker node ls | مشاهده نودهای موجود در Swarm |
| docker node ps <node-name> | مشاهده وظایف در حال اجرا روی یک نود |
| docker node promote <node> | ارتقای یک Worker به Manager |
| docker node demote <node> | تبدیل Manager به Worker |

**6. آپدیت Stack**

1. فایل docker-compose.yml را ویرایش کنید.
2. تغییرات را اعمال کنید:

bash

Copy

Download

docker stack deploy -c docker-compose.yml myapp

**7. حذف Stack**

bash

Copy

Download

docker stack rm myapp

**8. ویژگی‌های مخصوص Swarm در Compose**

**آپدیت بدون Downtime (Rolling Update)**

yaml

Copy

Download

deploy:

update\_config:

parallelism: 2 *# تعداد کانتینرهای آپدیت شده همزمان*

delay: 10s *# تأخیر بین آپدیت‌ها*

failure\_action: rollback *# بازگشت به نسخه قبلی در صورت خطا*

**محدودیت منابع (CPU/RAM)**

yaml

Copy

Download

deploy:

resources:

limits:

cpus: '0.50' *# 50% از یک هسته CPU*

memory: 512M *# حداکثر 512 مگابایت RAM*

**تعیین محل اجرای سرویس (Placement Constraints)**

yaml

Copy

Download

deploy:

placement:

constraints:

- node.role == manager *# فقط روی Manager اجرا شود*

**9. بهترین روش‌ها (Best Practices)**

✅ **استفاده از شبکه**overlay برای ارتباط بین چندین سرور.  
✅ **استفاده از Secrets** برای اطلاعات حساس (مثل پسورد دیتابیس).  
✅ **تعریف Health Check** برای بررسی سلامت سرویس‌ها.

مثال Health Check:

yaml

Copy

Download

healthcheck:

test: ["CMD", "curl", "-f", "http://localhost"]

interval: 30s

timeout: 10s

retries: 3

**10. عیب‌یابی (Troubleshooting)**

* مشاهده وظایف یک سرویس:

bash

Copy

Download

docker service ps myapp\_web

* بررسی وضعیت یک نود:

bash

Copy

Download

docker node inspect <node-name> --pretty

* خارج کردن یک نود از دسترس (برای تعمیرات):

bash

Copy

Download

docker node update --availability drain <node-name>

**جمع‌بندی**

* **Docker Compose** برای تعریف سرویس‌ها استفاده می‌شود.
* **Docker Swarm** برای مدیریت کانتینرها در چندین سرور مناسب است.
* **دستور**docker stack deploy برای اجرای docker-compose.yml در Swarm استفاده می‌شود.

**پیاده‌سازی Docker Swarm با Grafana, Node Exporter و Prometheus**

در این راهنما، یک Docker Swarm با ۳ سرویس زیر راه‌اندازی می‌کنیم:  
✅ 3 نمونه Grafana (برای مانیتورینگ  
✅ 3 نمونه Node Exporter (برای جمع‌آوری متریک‌های سیستم  
✅ 3 نمونه Prometheus (برای ذخیره‌سازی و پردازش داده‌های مانیتورینگ

1. فایل docker-compose.yml

version: '3.8'

services:

*# Prometheus (3 Replicas)*

prometheus:

image: docker.arvancloud.ir/prom/prometheus:latest

ports:

- "9090:9090"

volumes:

- prometheus-data:/prometheus

configs:

- source: prometheus\_config

target: /etc/prometheus/prometheus.yml

deploy:

replicas: 3

placement:

constraints:

- node.role == worker

node-exporter:

image: docker.arvancloud.ir/prom/node-exporter:latest

volumes:

- /proc:/host/proc:ro

- /sys:/host/sys:ro

- /:/rootfs:ro

command:

- '--path.procfs=/host/proc'

- '--path.sysfs=/host/sys'

- '--collector.filesystem.ignored-mount-points=^/(sys|proc|dev|host|etc)($$|/)'

deploy:

mode: global

resources:

limits:

memory: 256M

grafana:

image: docker.arvancloud.ir/grafana/grafana:latest

ports:

- "3000:3000"

volumes:

- grafana-data:/var/lib/grafana

environment:

- GF\_SECURITY\_ADMIN\_PASSWORD=admin123

deploy:

replicas: 3

update\_config:

parallelism: 1

delay: 10s

configs:

prometheus\_config:

file: ./prometheus.yml

volumes:

prometheus-data:

driver: local

grafana-data:

driver: local

networks:

default:

driver: overlay

2. فایل prometheus.yml

این فایل را در همان مسیر docker-compose.yml ذخیره کنید:

global:

scrape\_interval: 15s

scrape\_configs:

- job\_name: 'node-exporter'

static\_configs:

- targets: ['node-exporter:9100'] *# جمع‌آوری داده‌های Node Exporter*

- job\_name: 'prometheus'

static\_configs:

- targets: ['localhost:9090'] *# خود Prometheus*

3. راه‌اندازی سیستم

الف) تنظیم Docker Swarm

docker swarm init

ب) اجرای Stack

docker stack deploy -c docker-compose.yml monitoring

ج) بررسی وضعیت سرویس‌ها

docker service ls

4. دسترسی به سرویس‌ها

| سرویس | آدرس دسترسی |
| --- | --- |
| Prometheus | http://<SERVER\_IP>:9090 |
| Grafana | http://<SERVER\_IP>:3000 (ورود با admin/admin123) |
| Node Exporter | داده‌ها در Prometheus نمایش داده می‌شوند |

5. مقیاس‌گذاری (Scaling)

افزایش نمونه‌های Grafana

docker service scale monitoring\_grafana=5

کاهش نمونه‌های Prometheus

docker service scale monitoring\_prometheus=2

6. نکات مهم

1. Node Exporter به صورت mode: global اجرا می‌شود، یعنی روی هر نود یک نمونه وجود دارد.
2. Prometheus داده‌ها را در Volume ذخیره می‌کند، بنابراین با ریستارت از بین نمی‌رود.
3. Grafana از Volume برای ذخیره داشبوردها استفاده می‌کند.

7. عیب‌یابی

مشاهده لاگ‌ها

docker service logs monitoring\_prometheus

بررسی وضعیت نودها

docker node ls

حذف Stack در صورت نیاز

docker stack rm monitoring

**Core Components of Docker Swarm**

1. **Nodes**

* **Manager Nodes**
  + Control the cluster state and task scheduling
  + Use Raft consensus for leader election (needs odd number for HA)
  + Can also run containers (worker role by default)
* **Worker Nodes**
  + Execute tasks assigned by managers
  + Can be promoted to managers if needed

2. **Services**

* Define the desired state of containers:

bash

Copy

Download

docker service create --name web --replicas 3 nginx

* Types:
  + **Replicated services** (fixed number of identical containers)
  + **Global services** (one container per node)

3. **Tasks**

* A single container instance spawned by a service
* Swarm automatically replaces failed tasks

4. **Overlay Network**

* Virtual network spanning all swarm nodes
* Created with:

bash

Copy

Download

docker network create --driver overlay my-net

* Enables secure cross-node container communication

5. **Load Balancing**

* **Internal DNS**: Containers resolve services by name
* **Routing Mesh**:
  + Distributes incoming requests across replicas
  + Works even if no replica exists on the target node

6. **Secrets & Configs**

* **Secrets**: Encrypted sensitive data (e.g., passwords)

bash

Copy

Download

echo "db\_password" | docker secret create db\_pass -

* **Configs**: Non-sensitive configuration files

7. **Stacks**

* Multi-service deployments using compose files:

yaml

Copy

Download

*# docker-compose.yml*

version: '3.8'

services:

web:

image: nginx

deploy:

replicas: 3

* Deployed with:

bash

Copy

Download

docker stack deploy -c docker-compose.yml myapp

**Key Management Commands**

| **Command** | **Purpose** |
| --- | --- |
| docker swarm init | Initialize a swarm |
| docker swarm join | Join a worker/manager |
| docker node ls | List cluster nodes |
| docker service ls | List running services |
| docker service scale | Change replica count |

**Visualization**

Copy

Download

Manager Node

├── Orchestrates cluster

├── Maintains service state

└── Can run containers

Worker Node

└── Only executes tasks

Overlay Network

└── Connects all containers across nodes

Swarm is ideal for simple container orchestration with native Docker integration, while Kubernetes suits more complex scenarios. Would you like a practical deployment example?

* دستور docker commit

docker commit یک دستور در Docker است که تغییرات اعمال‌شده در یک کنتینر در حال اجرا را ذخیره کرده و از آن یک تصویر (image) جدید می‌سازد. این دستور مشابه این است که شما از حالت فعلی یک کنتینر (Snapshot) بگیرید و آن را به عنوان یک image جدید ذخیره کنید تا بعداً بتوانید از آن استفاده کنید.

#docker commit [container-id] new-image-name:v.

* **Buildx یک پلاگین برای Docker است که امکان ساخت تصاویر (images) را با قابلیت‌های پیشرفته‌تر نسبت به دستور سنتی docker build فراهم می‌کند. این ابزار به‌ویژه برای ساخت تصاویر چندپلتفرمی (multi-platform) و استفاده از قابلیت‌های BuildKit (موتور ساخت پیشرفته‌ی Docker) طراحی شده است.**
* **اسکن ایمیج داکری از لحاظ امنیتی با Trivy**

Installation

sudo apt-get install wget apt-transport-https gnupg lsb-release

wget -qO - https://aquasecurity.github.io/trivy-repo/deb/public.key | sudo apt-key add -

echo deb https://aquasecurity.github.io/trivy-repo/deb $(lsb\_release -sc) main | sudo tee -a /etc/apt/sources.list.d/trivy.list

sudo apt-get update

sudo apt-get install trivy

می تونید در ci-cd بزارید این trivy رو و بگید اگر تست ما آسیب پذیری high داشت رد نکنه و بیلد نشه.

دستورات

root@srv-01:~# trivy –help

#trivy image docker.arvancloud.ir/nginx

**GitLab**

* **Git, Gitlab**

پالیسی که شرکت ها دارن اینه برنامه نویس ها هر کدوم بک برنچ دارن.

root@srv-01:~# cd asghar

root@srv-01:~/asghar# git init

root@srv-01:~/asghar# git add file1

root@srv-01:~/asghar# git status

git config --global user.email "you@example.com"

git config --global user.name "Your Name"

root@srv-01:~/asghar# git config --global user.email “[sarlakhadi@gmail.com](mailto:sarlakhadi@gmail.com)”

root@srv-01:~/asghar# git config --global user.name "hadi"

root@srv-01:~/asghar# git branch blue

root@srv-01:~/asghar# git branch

blue

\* master

root@srv-01:~/asghar# git checkout blue

Switched to branch 'blue'

root@srv-01:~/asghar# git branch

\* blue

Master

root@srv-01:~/asghar# git log

root@srv-01:~/asghar# git checkout master

root@srv-01:~/asghar# git merge blue

* **Install Gitlab**

**نکته: حتما در گیت لب سازمانی از ldap active directort استفاده کنید. از لحاظ امنیتی مناسب تره حتی TFA هم میشه کرد. نکته دیگه اینکه یه یوزر که از سازمان میره باید کاربرشو بلاک کنید. امکان دیگه ای به نام band داریم که اگر کاربری تخلف کنه میتونید band ش کنیم تا تعین تکلیف بشه.**

**نکته : بعد از نصب Gitlab باید یک سری کار ها رو در ابتدا انجام بدین:**

**اولین کاری که باید انجام بدین باید URL برای گیت لب ست کنیم**: # vi /etc/gitlab/gitlab.rb

**بعد از اون باید دستور مقابل را برای gitlab reconfigure بزنید**  # gitlab-ctl reconfigure

**اتصال به LDAP رو انجام میدیم:**

* **تنظیمات GitLab برای اتصال به Active Directory (LDAP Integration)**

برای اتصال GitLab به **Active Directory (AD)** یا هر سرویس **LDAP** دیگر، باید تنظیمات مربوطه را در فایل پیکربندی GitLab (gitlab.rb) اعمال کنید. این تنظیمات امکان **احراز هویت کاربران** از طریق AD و همگام‌سازی گروه‌ها را فراهم می‌کند.

**🔰 مراحل اتصال GitLab به Active Directory**

**1. ویرایش فایل**gitlab.rb

فایل اصلی پیکربندی GitLab معمولاً در مسیر زیر قرار دارد:

bash

Copy

Download

sudo nano /etc/gitlab/gitlab.rb

**2. اضافه کردن تنظیمات LDAP/AD**

در این فایل، بخش مربوط به LDAP را پیدا کرده (یا اضافه کنید) و اطلاعات سرور Active Directory را وارد نمایید.  
**مثال:**

ruby

Copy

Download

gitlab\_rails['ldap\_enabled'] = true

gitlab\_rails['ldap\_servers'] = {

'main' => {

'label' => 'LDAP',

'host' => 'ad.example.com', *# آدرس سرور AD/LDAP*

'port' => 389, *# پورت LDAP (معمولاً 389 یا 636 برای SSL)*

'uid' => 'sAMAccountName', *# فیلد نام کاربری در AD (معمولاً sAMAccountName)*

'bind\_dn' => 'CN=GitLab\_User,CN=Users,DC=example,DC=com', *# کاربر اتصال به AD*

'password' => 'password123', *# رمز عبور کاربر Bind DN*

'encryption' => 'plain', *# یا 'start\_tls' یا 'simple\_tls' برای رمزنگاری*

'verify\_certificates' => false, *# اگر از SSL استفاده می‌کنید، بررسی گواهی*

'active\_directory' => true, *# فعال برای AD (مخصوص Active Directory)*

'base' => 'DC=example,DC=com', *# DN پایه برای جستجو*

'user\_filter' => '(memberOf=CN=GitLab\_Users,CN=Users,DC=example,DC=com)' *# فیلتر اختیاری*

}

}

**3. ذخیره و اعمال تغییرات**

پس از ویرایش فایل، دستور زیر را اجرا کنید تا تغییرات اعمال شوند:

bash

Copy

Download

sudo gitlab-ctl reconfigure

**4. بررسی اتصال (اختیاری)**

برای تست اتصال به LDAP، از دستور زیر استفاده کنید:

bash

Copy

Download

sudo gitlab-rake gitlab:ldap:check

**⚙ تنظیمات پیشرفته (اختیاری)**

**🔹 محدود کردن دسترسی به گروه‌های خاص**

اگر می‌خواهید فقط کاربران عضو یک گروه خاص در AD بتوانند وارد GitLab شوند، از user\_filter استفاده کنید:

ruby

Copy

Download

user\_filter: '(&(objectCategory=Person)(memberOf=CN=GitLab\_Access,OU=Groups,DC=example,DC=com))'

**🔹 همگام‌سازی گروه‌های AD با GitLab**

برای همگام‌سازی گروه‌های AD با GitLab، در تنظیمات گروه در GitLab، گزینه **LDAP Synchronization** را فعال کنید.

**🔹 استفاده از LDAPS (SSL)**

اگر از **LDAP روی پورت 636 (LDAPS)** استفاده می‌کنید، تنظیمات زیر را اعمال کنید:

ruby

Copy

Download

encryption: 'simple\_tls',

port: 636,

verify\_certificates: true *# اگر گواهی معتبر دارید*

**🔍 عیب‌یابی مشکلات رایج**

| **مشکل** | **راه‌حل** |
| --- | --- |
| اتصال برقرار نمی‌شود | بررسی host، port و bind\_dn + اطمینان از دسترسی شبکه به سرور AD. |
| کاربران پیدا نمی‌شوند | بررسی base و user\_filter + تست با ابزار ldapsearch. |
| خطای گواهی SSL | اگر از خودامضا (Self-Signed) استفاده می‌کنید، verify\_certificates: false کنید. |
| همگام‌سازی گروه‌ها کار نمی‌کند | اطمینان از صحت memberOf و ساختار گروه در AD. |

**🎯 نتیجه‌گیری**

* GitLab از **LDAP/Active Directory** پشتیبانی کامل می‌کند.
* تنظیمات اصلی در gitlab.rb انجام می‌شود.
* پس از اعمال تغییرات، حتماً gitlab-ctl reconfigure را اجرا کنید.
* برای محیط‌های Enterprise، **LDAPS** توصیه می‌شود.

**تنظیمات GitLab برای اتصال به Jira (اتصال یکپارچه‌سازی - Integration)**

اتصال GitLab به Jira امکان **ردیابی خودکار Issueها، همگام‌سازی کامیت‌ها و نمایش اطلاعات توسعه در Jira** را فراهم می‌کند. این اتصال از طریق **Jira Integration** یا **API** انجام می‌شود.

**🔰 روش‌های اتصال GitLab به Jira**

**1. استفاده از Jira Integration داخلی GitLab (ساده‌ترین روش)**

این روش نیازی به کدنویسی ندارد و از طریق تنظیمات پروژه در GitLab انجام می‌شود.

**مراحل تنظیم:**

1. **در GitLab:**
   * به پروژه مورد نظر بروید.
   * از منوی سمت چپ، Settings > Integrations را انتخاب کنید.
   * در لیست Integrations، Jira را پیدا و کلیک کنید.
2. **تنظیمات Jira Integration:**
   * **Jira site URL:** آدرس سرور Jira (مثلاً https://your-company.atlassian.net)
   * **Username/Email:** ایمیل ادمین Jira
   * **Password/API token:** [API Token از Jira](https://id.atlassian.com/manage-profile/security/api-tokens) (به جای رمز عبور استفاده شود)
   * **Project Key:** کلید پروژه Jira (مثلاً PROJ)
   * **Enable comments:** فعال کردن کامنت‌های خودکار
   * **Enable transitions:** تغییر وضعیت خودکار Issueها بر اساس کامیت‌ها
3. **ذخیره و فعال‌سازی:**
   * تنظیمات را ذخیره کنید.
   * اکنون می‌توانید Issueهای Jira را در Merge Requestها و کامیت‌های GitLab مشاهده کنید.

**2. استفاده از Webhook (برای همگام‌سازی پیشرفته)**

اگر نیاز به **همگام‌سازی سفارشی** دارید (مثلاً تغییر وضعیت خودکار بر اساس کامیت‌ها)، می‌توانید از Webhook استفاده کنید.

**مراحل تنظیم Webhook در GitLab:**

1. **در Jira:**
   * به Settings > System > Webhooks بروید.
   * یک Webhook جدید ایجاد کنید و URL زیر را وارد کنید:

Copy

Download

https://your-gitlab-instance.com/api/v4/projects/:id/jira/transitions

* + رویدادهای مورد نظر (مثلاً Merge Request یا Pipeline Success) را انتخاب کنید.

1. **در GitLab:**
   * در تنظیمات پروژه، به Settings > Webhooks بروید.
   * URL سرور Jira را وارد کنید (مثلاً https://jira.example.com/rest/api/2/issue/{issue.key}/transitions).
   * رویدادهای مورد نظر (مثلاً Push events یا Merge Request events) را انتخاب کنید.

**3. استفاده از API (برای توسعه‌دهندگان)**

اگر نیاز به **یکپارچه‌سازی پیشرفته** دارید، می‌توانید از **Jira REST API** و **GitLab API** استفاده کنید.

**مثال: تغییر وضعیت Jira Issue با کامیت GitLab**

bash

Copy

Download

*# مثال با cURL*

curl -X POST \

-H "Authorization: Bearer YOUR\_JIRA\_API\_TOKEN" \

-H "Content-Type: application/json" \

-d '{"transition":{"id":"31"}}' \

"https://your-company.atlassian.net/rest/api/3/issue/ISSUE-123/transitions"

**پیکربندی در**.gitlab-ci.yml**برای خودکارسازی:**

yaml

Copy

Download

jira-transition:

stage: deploy

script:

- |

curl -X POST \

-H "Authorization: Bearer $JIRA\_TOKEN" \

-H "Content-Type: application/json" \

-d '{"transition":{"id":"31"}}' \

"https://jira.example.com/rest/api/2/issue/$JIRA\_ISSUE/transitions"

only:

- main

**🔍 عیب‌یابی مشکلات رایج**

| **مشکل** | **راه‌حل** |
| --- | --- |
| **اتصال برقرار نمی‌شود** | بررسی URL، API Token و دسترسی شبکه. |
| **کامیت‌ها در Jira نمایش داده نمی‌شوند** | بررسی Project Key و فعال بودن قابلیت Commit Mentions در Jira. |
| **Webhook کار نمی‌کند** | بررسی لاگ‌های GitLab (/var/log/gitlab) و Jira. |
| **خطای 403 (دسترسی نداری)** | اطمینان از صحت API Token و مجوزهای کاربر در Jira. |

**🎯 بهترین روش‌ها برای یکپارچه‌سازی GitLab و Jira**

✅ از **API Token** به جای رمز عبور استفاده کنید.  
✅ در صورت نیاز به همگام‌سازی پیشرفته، از **Webhook + CI/CD** استفاده کنید.  
✅ برای ردیابی خودکار Issueها، حتماً **کلید پروژه (Project Key)** را در کامیت‌ها ذکر کنید (مثلاً PROJ-123).  
✅ لاگ‌های GitLab و Jira را برای خطاها بررسی کنید.

* **فرق بین گروه و پروژه در گیت لب:**

در GitLab، دو مفهوم گروه (Group) و پروژه (Project) وجود دارند که هرکدام کاربرد متفاوتی دارند:

🔹 **گروه (Group)**

* یک فضای سازمانی برای دسته‌بندی پروژه‌ها و مدیریت دسترسی‌هاست.
* می‌تواند شامل چندین پروژه، زیرگروه (Subgroup) و اعضا (Members) باشد.
* کاربرد اصلی:
  + مدیریت دسترسی متمرکز (مثلاً تعیین نقش‌های Developer، Maintainer برای کل گروه)
  + سازمان‌دهی پروژه‌های مرتبط (مثلاً گروه Backend شامل پروژه‌های API، Microservices و...).
  + اشتراک‌گذاری متغیرهای CI/CD در سطح گروه.
  + مدیریت مالی و گزارش‌گیری برای چندین پروژه.

**🔹 پروژه (Project)**

* معادل یک ریپازیتوری Git است که شامل کد، issues، CI/CD pipelines و... می‌شود.
* کاربرد اصلی:
  + مدیریت کد منبع (مانند یک پروژه جداگانه برای Frontend یا Backend).
  + پیاده‌سازی CI/CD (تعریف .gitlab-ci.yml برای اتوماسیون تست و deploy).
  + مدیریت Issues، Merge Requests و Wiki.

🔸 مقایسه کلیدی

| **ویژگی** | **گروه (Group)** | **پروژه (Project)** |
| --- | --- | --- |
| ساختار | شامل چندین پروژه/زیرگروه | یک ریپازیتوری مستقل |
| دسترسی | مدیریت دسترسی گروهی | تنظیم دسترسی اختصاصی |
| CI/CD | تعریف متغیرهای مشترک | تنظیم pipeline مخصوص پروژه |
| کاربرد | سازمان‌دهی پروژه‌ها | توسعه و مدیریت کد |

مثال:

* یک شرکت ممکن است یک گروه به نام Company داشته باشد که زیرگروه‌های Frontend و Backend دارد.
* درون Frontend، چندین پروژه مثل Web-App و Mobile-App وجود دارد.
* اعضای گروه Frontend به همه پروژه‌های آن دسترسی دارند، اما تنظیمات CI/CD هر پروژه مستقل است.

💡 جمع‌بندی:

* اگر نیاز به سازمان‌دهی چندین پروژه دارید، از گروه استفاده کنید.
* اگر می‌خواهید کد یک محصول خاص را مدیریت کنید، پروژه ایجاد کنید.

نکته: توی شرکت هاتون معمعمولا گروه ها رو private تعریف کنید.

نکته: اگر علامت آچار پایین صفحتون باشه شما ادمینید و بقیه این رو ندارن

نکته: غیر از شما که devops engineer هستید بقیه بوزر regular تعریف می شوند.

Block: وقتی یک کاربری از شرکت میره باید کاربری اون رو بلاک کنید

Ban: وقتی یک کاربر موقتا دسترسیش قطع بشه اونو ban می کنید.

نکته: در محیط product معمولا یوزر ها رو تک تک تعریف نمیکنیم و گیت لب رو به اکتیو دایرکتوری وصل می کنیم. با یوزر پسورد اکتیو دایرکتوری لاکین میکنه و بعد اونجا براش پرمیشن بدیم.

نکته: وقتی یه پروژه رو میسازین اولین commit رو میزنه و اولین کامیت readme هست.

**🔰 انواع نقش‌ها (Roles) در GitLab**

**۱. نقش‌های اصلی در پروژه (Project Roles)**

این نقش‌ها برای **تک پروژه** اعمال می‌شوند:

| **نقش (Role)** | **دسترسی‌ها (Permissions)** |
| --- | --- |
| **Guest** | 👀 فقط مشاهده پروژه (بدون ویرایش) |
| **Reporter** | 🔍 مشاهده + گزارش Issues و Merge Requests |
| **Developer** | 💻 Commit زدن، ایجاد MR، مدیریت Issues |
| **Maintainer** | ⚙️ مدیریت پروژه، تنظیم CI/CD، حذف Branchها |
| **Owner** | 👑 تمام دسترسی‌ها + حذف پروژه |

**۲. نقش‌های اصلی در گروه (Group Roles)**

این نقش‌ها برای **کل گروه و زیرگروه‌ها** اعمال می‌شوند:

| **نقش (Role)** | **دسترسی‌ها (Permissions)** |
| --- | --- |
| **Guest** | 👀 فقط مشاهده گروه و پروژه‌ها |
| **Reporter** | 🔍 مشاهده + گزارش Issues |
| **Developer** | 💻 مشارکت در پروژه‌های گروه |
| **Maintainer** | ⚙️ مدیریت پروژه‌ها و تنظیمات گروه |
| **Owner** | 👑 تمام دسترسی‌ها + حذف گروه |

**۳. نقش Instance Admin (مدیر سرور GitLab)**

* فقط در نسخه **Self-Managed** (نصب شده روی سرور داخلی) وجود دارد.
* دسترسی به **همه گروه‌ها و پروژه‌ها** حتی Private.
* می‌تواند **کاربران جدید ایجاد کند** و **تنظیمات کلی GitLab** را تغییر دهد.

**🔸 تفاوت نقش‌های Project vs. Group**

* اگر کاربری در **گروه** نقش **Maintainer** داشته باشد، می‌تواند **پروژه‌های داخل آن گروه** را مدیریت کند.
* اما اگر فقط در یک **پروژه خاص** نقش **Developer** داشته باشد، نمی‌تواند تنظیمات کلی گروه را تغییر دهد.

**🎯 نکات مهم**

✅ **ارث‌بری نقش‌ها**: اگر کاربری در یک **گروه** نقش داشته باشد، این نقش به **پروژه‌های داخل آن گروه** هم اعمال می‌شود (مگر اینکه دسترسی جداگانه تعریف شود).  
✅ **حداقل سطح دسترسی**: می‌توان در تنظیمات گروه، حداقل نقش مورد نیاز برای ایجاد پروژه را تعیین کرد.  
✅ **Custom Roles** (در نسخه **Premium/Gold**): می‌توان نقش‌های سفارشی ساخت.