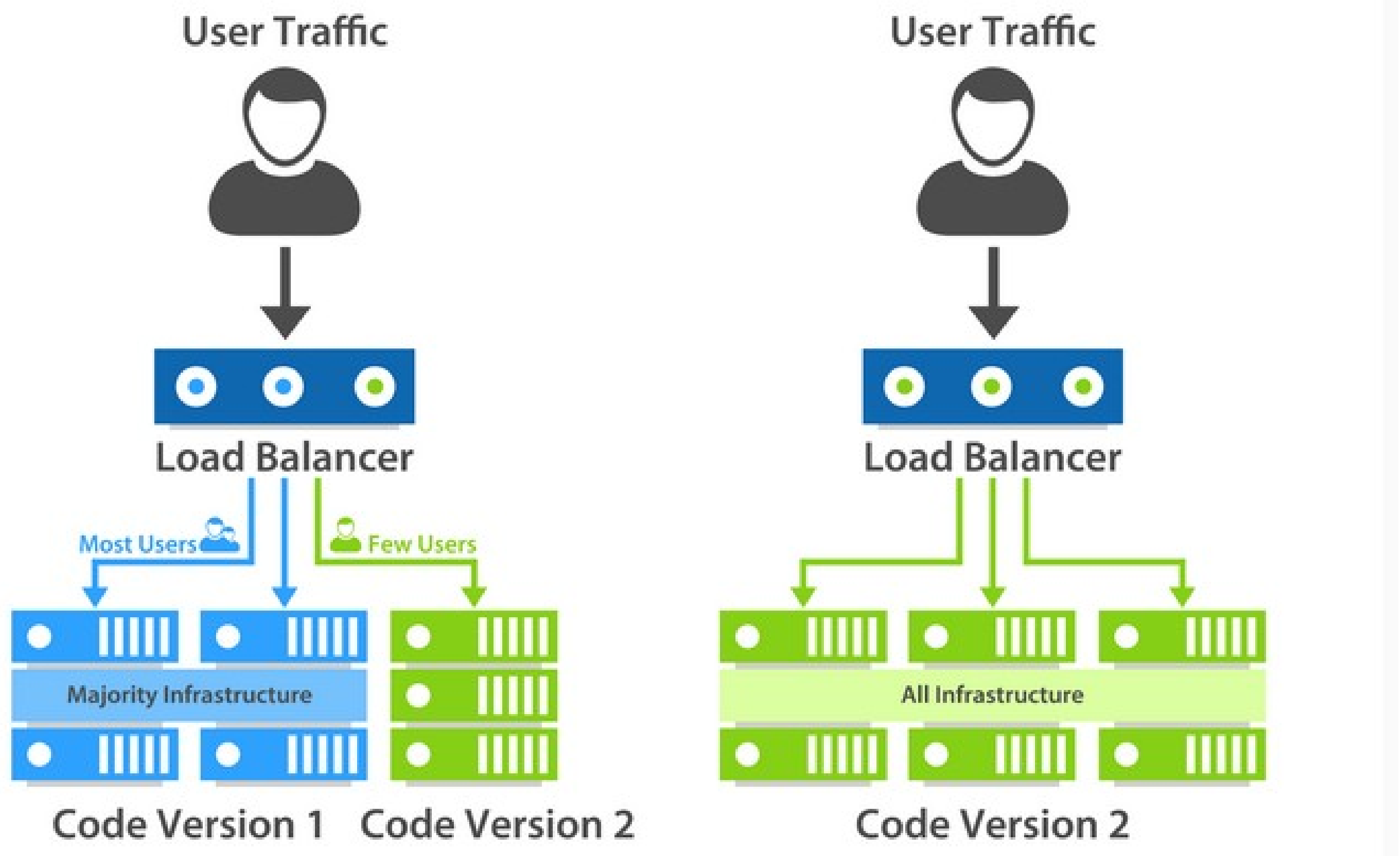
Docker compose

Why we need orchestration

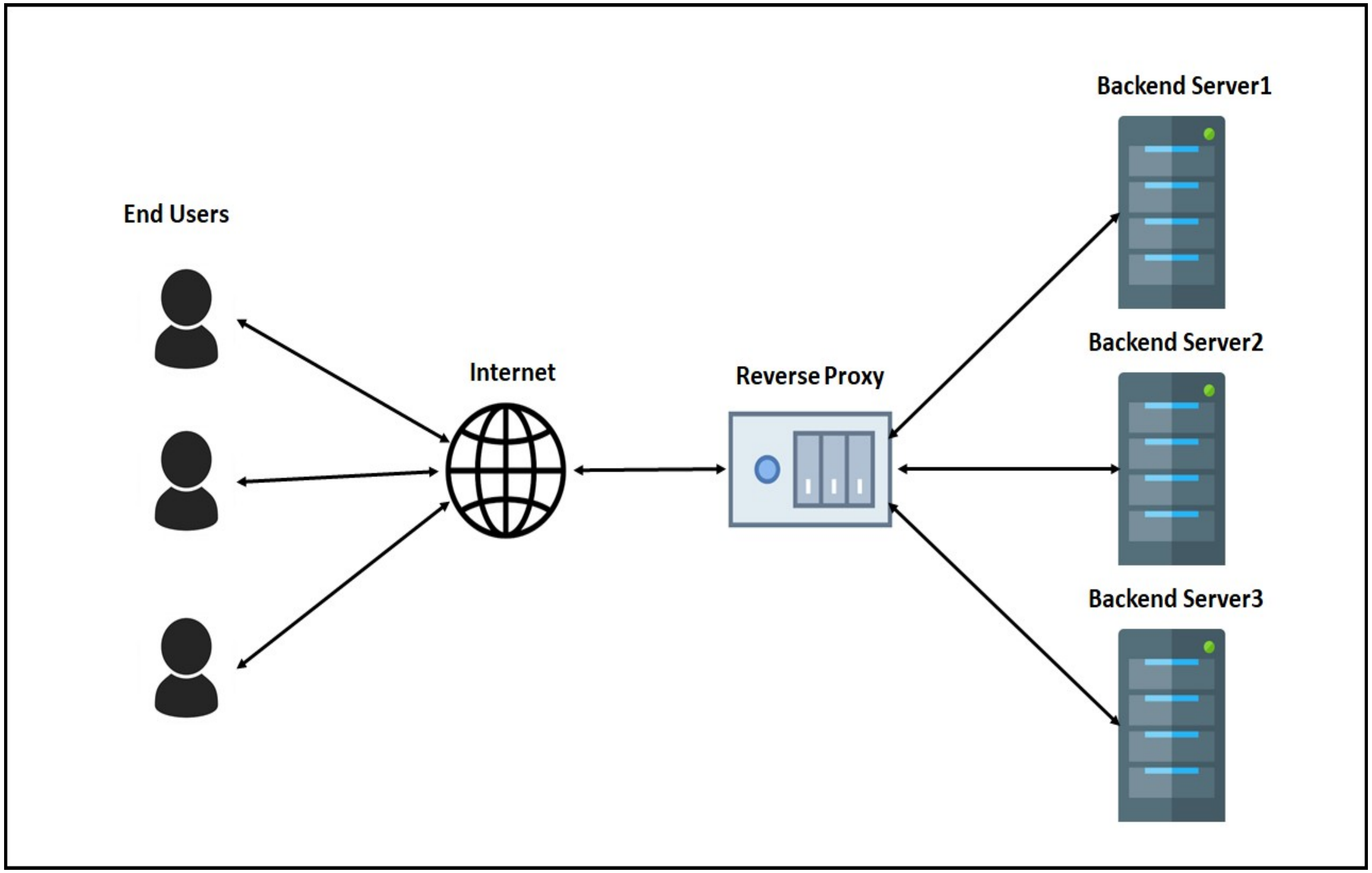
- Run an ecosystem of applications
- Prevent downtime during deployments
- Networking and routing capabilities
- Scalability capabilities

Deployments downtime



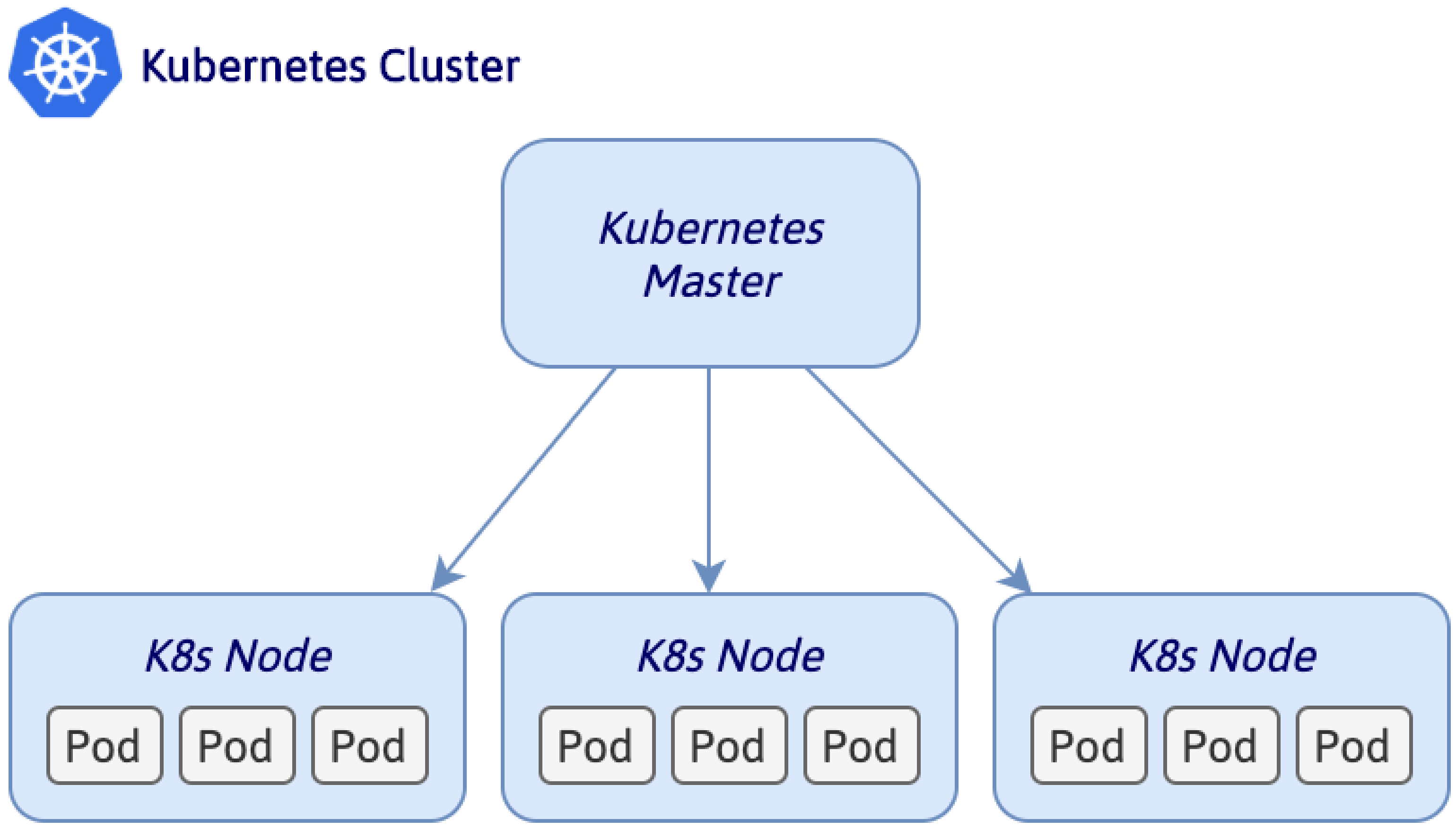
Networking / routing

- K8s offer a wide range of routing possibilities es. Nginx, traefik, calico, istio



Scalability

- K8s clusters are composed by nodes. A pool of resources (CPU / RAM) is than shared across pods. A Pod is a set of containers



Orchestration basics

- **Docker-compose**, docker swarm
- Kubernetes, minikube, microk8s
- Openshift (Paas)
- Rancher (hybrid cloud)
- Others: Mesos, Nomad

Orchestration basics

Docker compose has limited orchestration capabilities:

- use **depends_on** to define start priorities
- **Restart policy**
- **volumes** and **network**
- Basically is a transposition of the docker syntax to a yaml file

Volumes declaration syntax

- **host-mounted** volumes

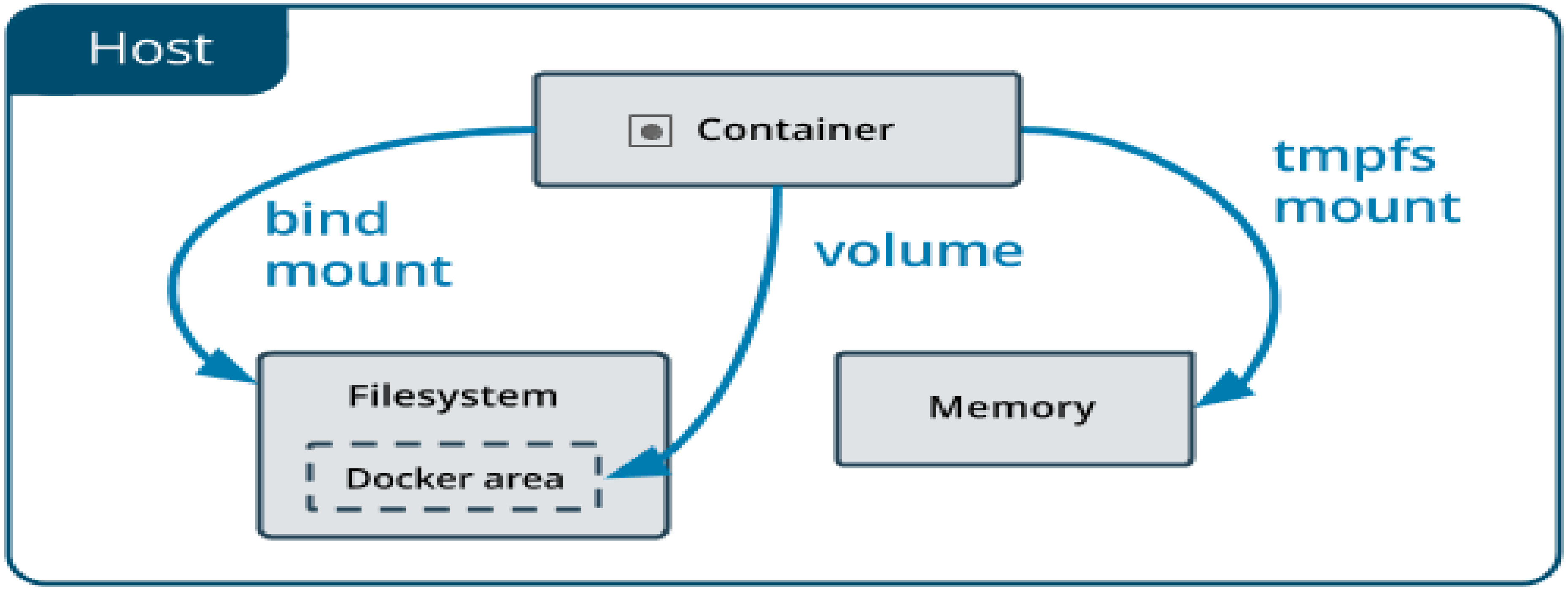
  /host/path:/container/path
- **named** volumes

  volume_name:/container/path
- **shared** volumes

  --volumes-from container_name

Volumes mount types

- Volumes are stored in a part of the host filesystem which is managed by Docker (/var/lib/docker/volumes/ on Linux). Non-Docker processes should not modify this part of the filesystem. Volumes are the best way to persist data in Docker.
- Bind mounts may be stored anywhere on the host system. They may even be important system files or directories. Non-Docker processes on the Docker host or a Docker container can modify them at any time.
- tmpfs mounts are stored in the host system's memory only, and are never written to the host system's filesystem.



Networking

netstat -rn
ifconfig

```
aureliano@aureliano-N141CU ~/git/devops-B3-2023 (main)$ netstat -rn
Kernel IP routing table
Destination     Gateway         Genmask         Flags   MSS Window  irtt Iface
0.0.0.0         192.168.1.254   0.0.0.0         UG        0 0          0 wlp0s20f3
10.0.3.0        0.0.0.0         255.255.255.0   U         0 0          0 lxcbr0
169.254.0.0     0.0.0.0         255.255.0.0     U         0 0          0 wlp0s20f3
172.17.0.0      0.0.0.0         255.255.0.0     U         0 0          0 docker0
```

Can customize the ip range of docker.

```
Docker network ls
Docker network inspect my_network
Cat /etc/docker/daemon.json
```

Frequent use case: I want to connect from a container to a service on the host

The host has a changing IP address, or none if you have no network access. We recommend that you connect to the special DNS name host.docker.internal, which resolves to the internal IP address used by the host.

You can also reach the gateway using gateway.docker.internal (**By default, Docker uses 172.17.0.0/16.**)

Network declaration syntax

Can create subnets. This enforce isolation between containers

```
version: '3.6'services:
  api-gateway:
    container_name: api-gateway
    image: api-gateway
    networks:
      - gateway
    ports:
      - 9090:8080
    restart:
      on-failure
  api-gateway-replica:
    container_name: api-gateway-
replica
    image: api-gateway
    networks:
      - gateway-replica
    ports:
      - 9092:8080
    restart:
      on-failure
networks:
  gateway: {}
  gateway-replica: {}
```

Can customize the ip range of docker.

```
Docker network ls
Docker network inspect my_network
Cat /etc/docker/daemon.json
```

Orchestration – docker compose

Compose is a tool for defining and running multi-container Docker applications. With Compose, you use a YAML file to configure your application's services. Then, with a single command, you create and start all the services from your configuration.

It also has commands for managing the whole lifecycle of your application:

- Start, stop, and rebuild services
- View the status of running services
- Stream the log output of running services
- Run a one-off command on a service

The key features of Compose that make it effective are:

- Have multiple isolated environments on a single host
- Preserve volume data when containers are created
- Only recreate containers that have changed
- Support variables and moving a composition between environments

Orchestration – docker compose

Start postgres and redis: **docker-compose up**

```yaml
version: "3.8"
services:
  redis:
    image: redis:7-bullseye
    container_name: redis-airflow
    restart: always
    command: ["--databases", "1"]
    healthcheck:
      test: ["CMD", "redis-cli", "ping"]
      interval: 10s
      timeout: 5s
      retries: 5
    ports:
      - 6379:6379
    volumes:
      - redisdata:/data

  postgres:
    image: postgres:14.7-bullseye
    container_name: postgres
    volumes:
      - pgdata_mlflow:/var/lib/postgresql/data/pg_data
      - ./docker/postgresql/scripts:/var/lib/postgresql/scripts
    restart: always
    environment:
      - POSTGRES_PORT=5432
      - POSTGRES_PASSWORD=ynov
      - POSTGRES_USER=ynovcampus
      - POSTGRES_DB=ynov
      - PGDATA=/var/lib/postgresql/data/pg_data

volumes:
  pgvol:
  backup_pg_airflow:
```

Expose container port to host port

Host mounted volume declaration

Env var declaration

Named volumes declaration

Orchestration – docker compose

```
version: "3.8"
services:
  redis:
    build:
      context: ./my/context
      dockerfile: ./my/Dockerfile
    container_name: redis-airflow
    restart: always
    command: ["--databases", "1"]
    healthcheck:
      test: ["CMD", "redis-cli", "ping"]
      interval: 10s
      timeout: 5s
      retries: 5
...
```

Define the root (context) from wich files are resolved by dockerfile

Define the dockerfile if not at same level of docker-compose.yaml

build redis: **docker-compose build redis**

ENVIRONMENT VARIABLES

- DOTENV: env variables can be moved to .env files
- Another way is to set env var as SYSTEM or USER variables
- Set default ENV in the Dockerfile
- Override ENV at runtime via e flag or in the dockerfile

DOCKER MAINTENANCE

- Inspect: ls /var/lib/docker/containers
- docker image prune
- docker container prune
- docker network prune
- docker volume prune
- docker system prune