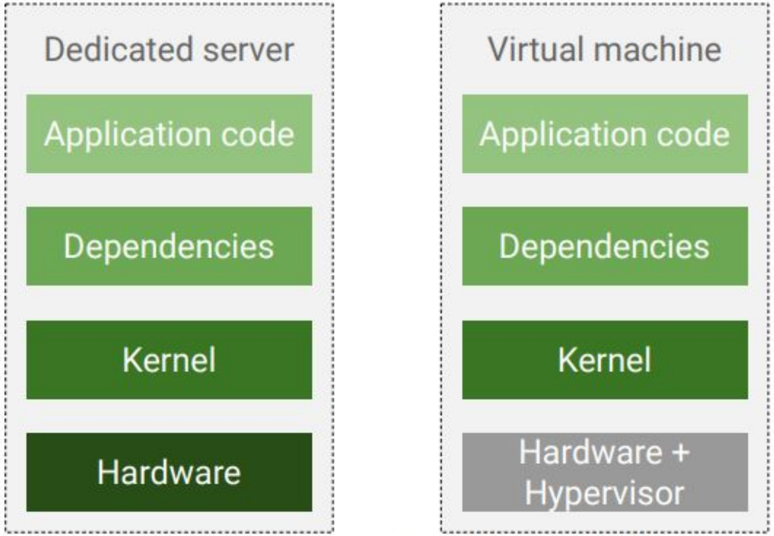
Build a simple web application with Docker

Everything about a service is in one place in git, and managed by one process.

A simple web application using Python Flask

```
from flask import Flask, Response
app = Flask(__name__)

@app.route("/")
def hello():
    return Response("Hello World")

if __name__ == "__main__":
    app.run("0.0.0.0", port=5000, debug=True)
```

Hypervisors create and manage virtual machines



Dedicated server

Application code

Dependencies

Kernel

Hardware

Virtual machine

Application code

Dependencies

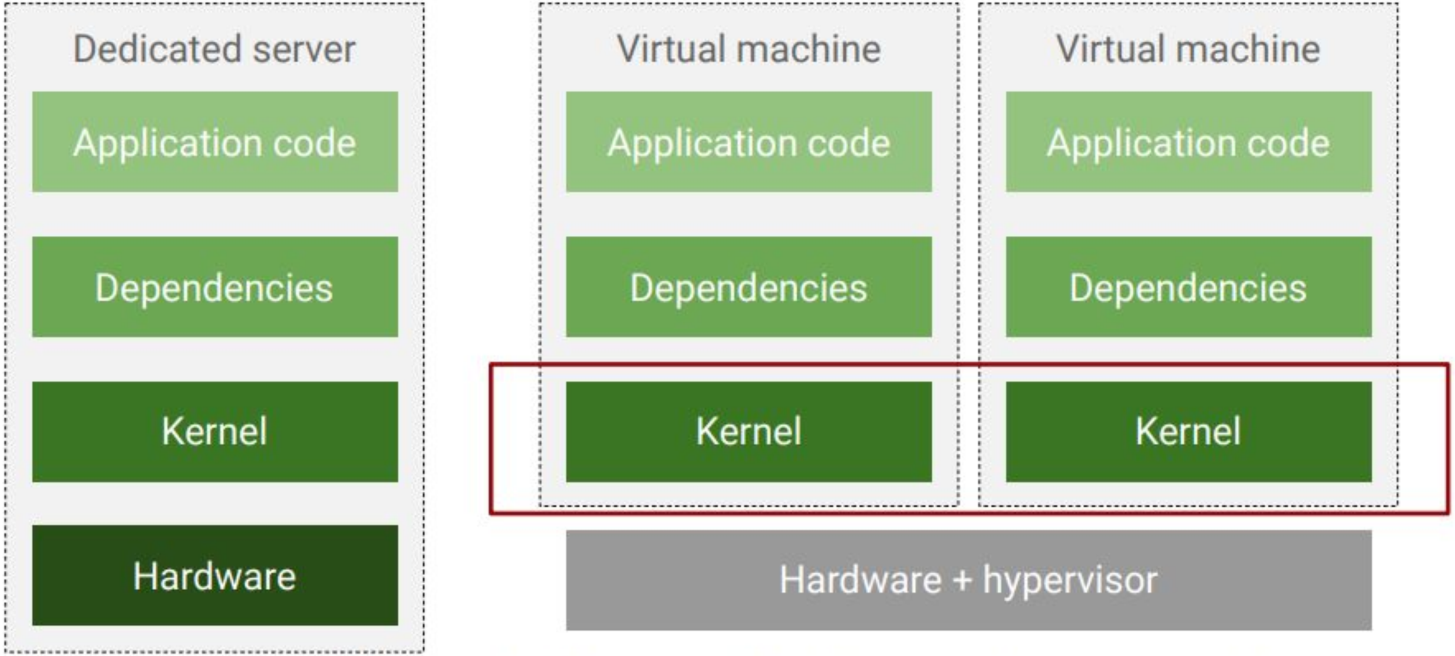
Kernel

Hardware + Hypervisor

Deployment ~months
Low utilization
Not portable

Deployment ~days (mins)
Improved utilization
Hypervisor-specific

The VM-centric way to solve this problem



Dedicated server

Application code

Dependencies

Kernel

Hardware

Virtual machine

Application code

Dependencies

Kernel

Virtual machine

Application code

Dependencies

Kernel

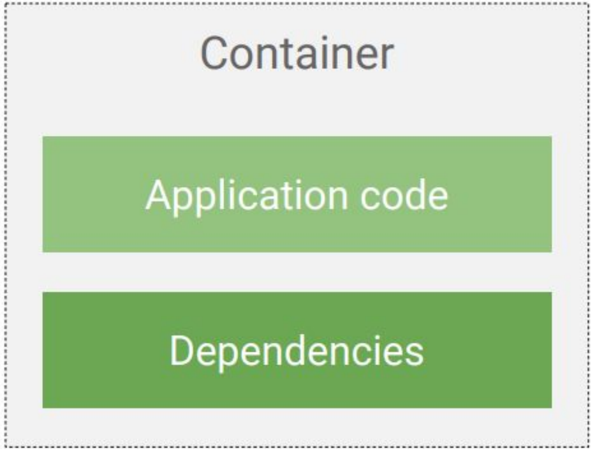
Hardware + hypervisor

Deployment ~months
Not portable
Low utilization

Deployment ~days (mins)
Hypervisor-specific
Low isolation; tied to OS

Deployment ~days (mins)
Hypervisor-specific
Redundant OS

Containers are lightweight, standalone, resource-efficient, portable, executable packages

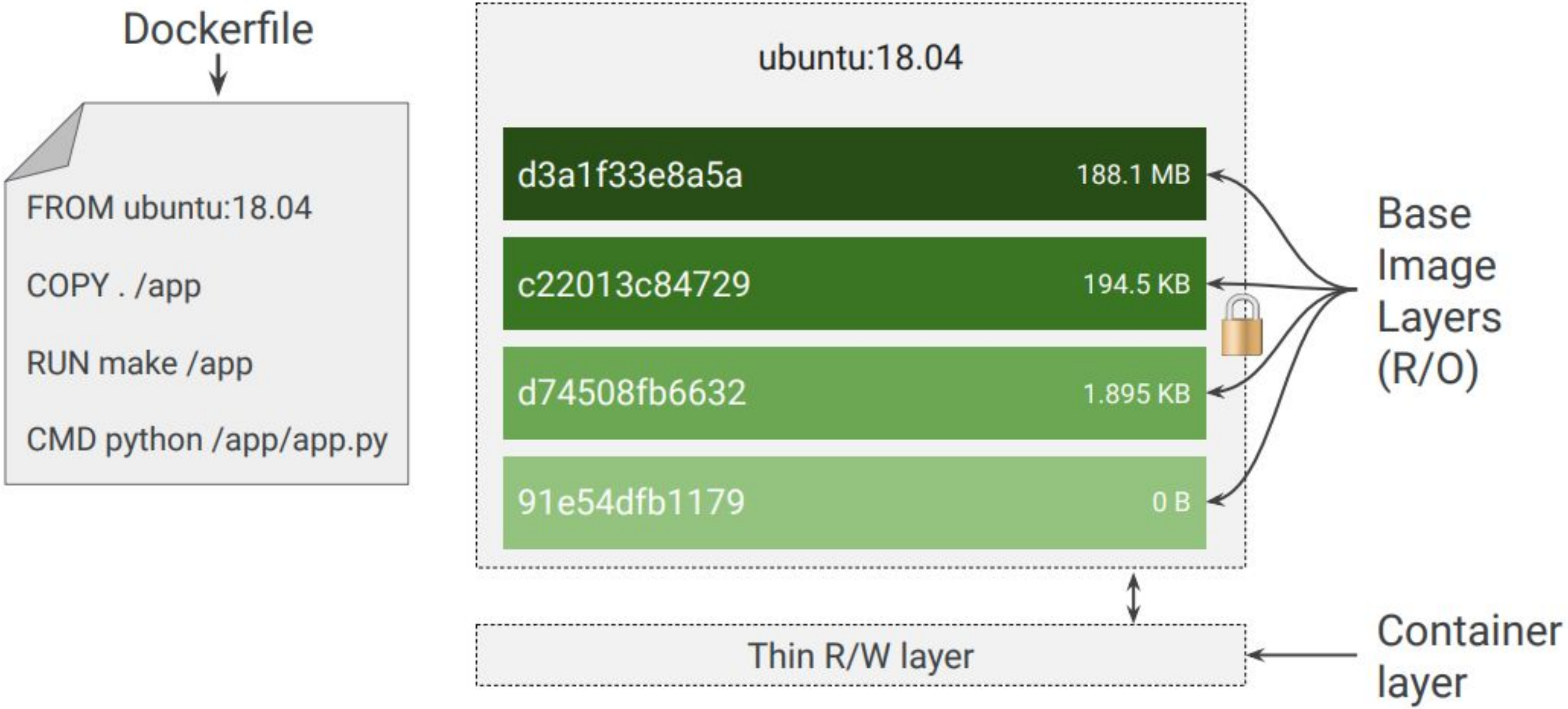


Container

Application code

Dependencies

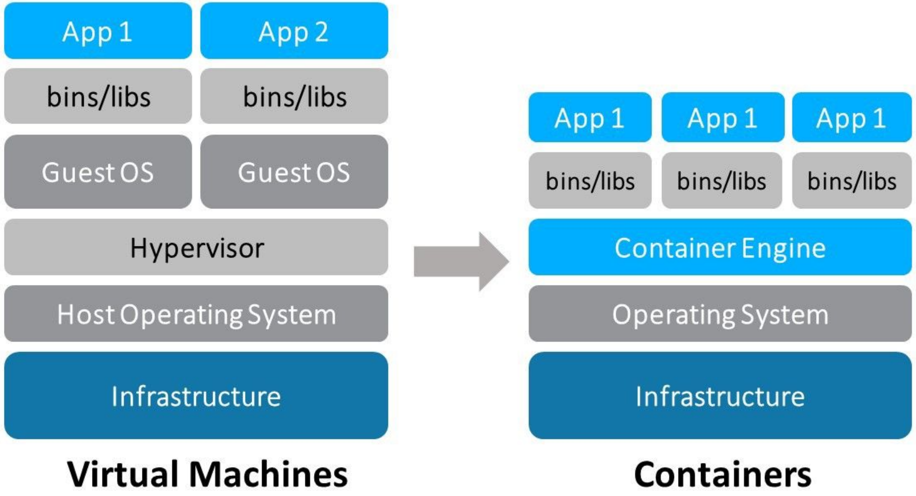
Containers are structured in layers



```
FROM ubuntu:18.04

COPY . /app

RUN make /app

CMD python /app/app.py
```

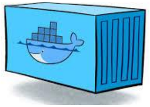
What is Container?

A container, unlike a virtual machine, does not require or include a separate operating system. Instead, it relies on the kernel's functionality and uses resource isolation for CPU and memory, and separate namespaces to isolate the application's view of the operating system. Docker is the most famous container.

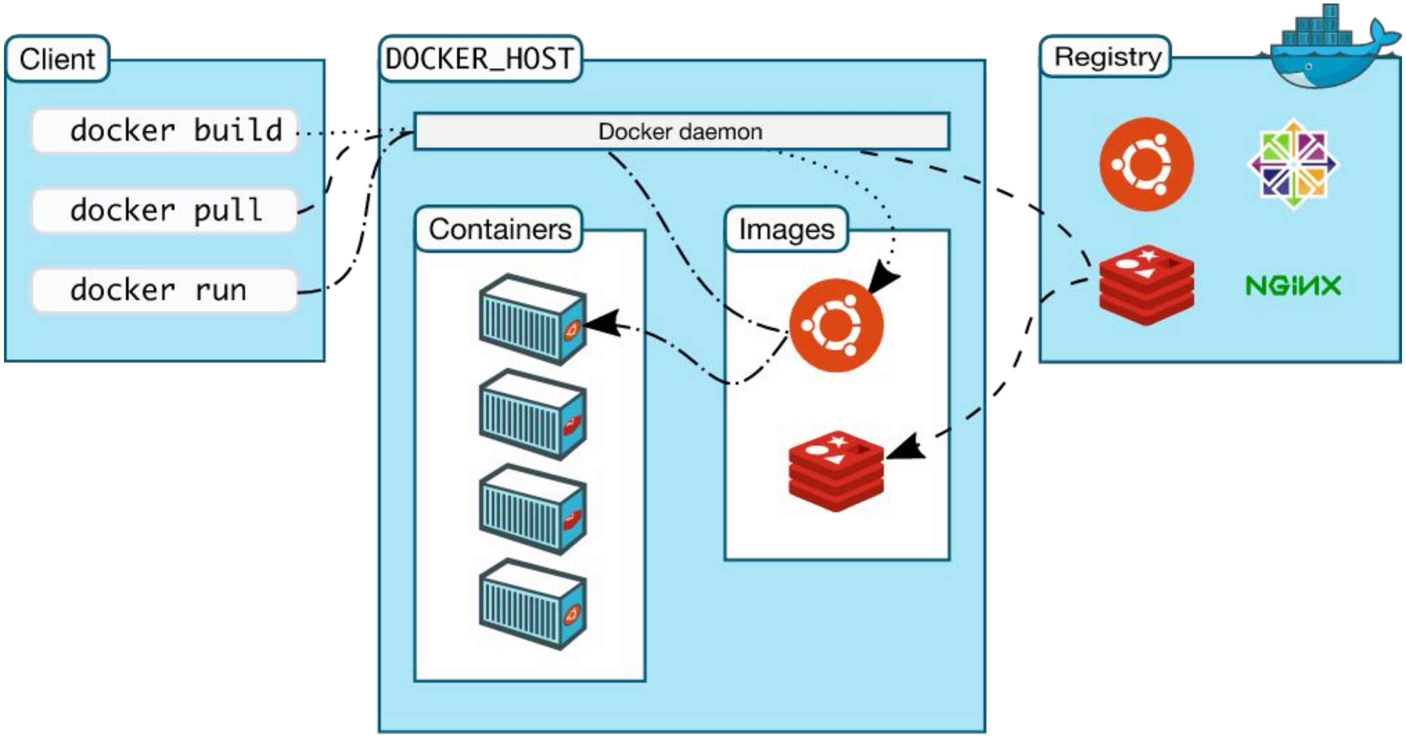


Docker Basics

- Docker Images
  - Contains everything needed to run an application - all dependencies, configuration, scripts, binaries, etc
  - Collection of files
  - Images stored in registries
  - Defined by a Dockerfile
- Docker Containers
  - A running instance (process) of an image pulled from a registry.
  - Private environment (process namespace, filesystem)
- Docker Registry
  - https://hub.docker.com/

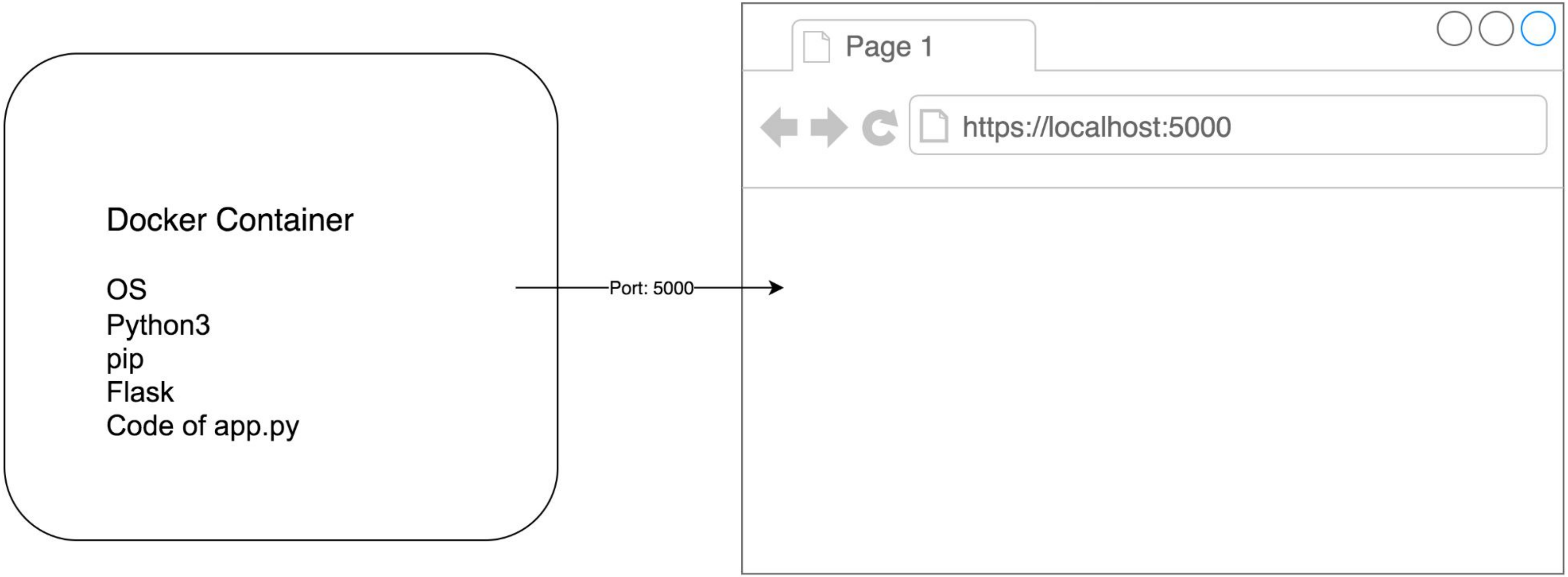


Docker Architecture



Dockerfile: builds an docker image

```
FROM python:3
RUN mkdir /app
WORKDIR /app
ADD . /app
RUN pip install --trusted-host pypi.python.org -r requirements.txt
EXPOSE 5000
CMD ["python", "app.py"]
```

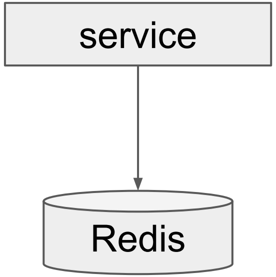


Docker Container

OS
Python3
pip
Flask
Code of app.py

A web application with Redis

- API
  - /set: store key-value pair
  - /get: retrieves value of one key
  - /: returns hello world information



Redis

- In-memory data structure store
- Redis is an advanced **key-value** store, where keys can contain data structures such as strings, hashes, lists, sets, and sorted sets. Supporting a set of **atomic operations** on these data types.
- Blazing fast
  - Benchmark: 5M/s on AWS c5.18xlarge with 72 threads
- Can be used as Database, a Caching layer or a Message broker.

Redis can persist data to the disk

Redis is fast

Not only key-value store

FYI

- Official webpage: http://redis.io
- Source Code: https://github.com/antirez/redis
- Online Demo: http://try.redis.io

Use cases

- Redis is **NOT** a replacement for Relational Databases nor Document Stores.
- It might be used complementary to a SQL relational store, and/or NoSQL document store.
- Best Used: For rapidly changing data with a foreseeable database size (should fit mostly in memory).

Redis Data Structures

- **String**: Binary-safe strings.
- **Lists**: collections of string elements sorted according to the order of insertion. They are basically linked lists.
- **Sets**: collections of unique, unsorted string elements.
- **Sorted sets**: similar to Sets but where every string element is associated to a floating number value, called score. The elements are always taken sorted by their score, so unlike Sets it is possible to retrieve a range of elements (for example you may ask: give me the top 10, or the bottom 10).
- **Hashes**: which are maps composed of fields associated with values. Both the field and the value are strings. This is very similar to Ruby or Python hashes.

Example 1

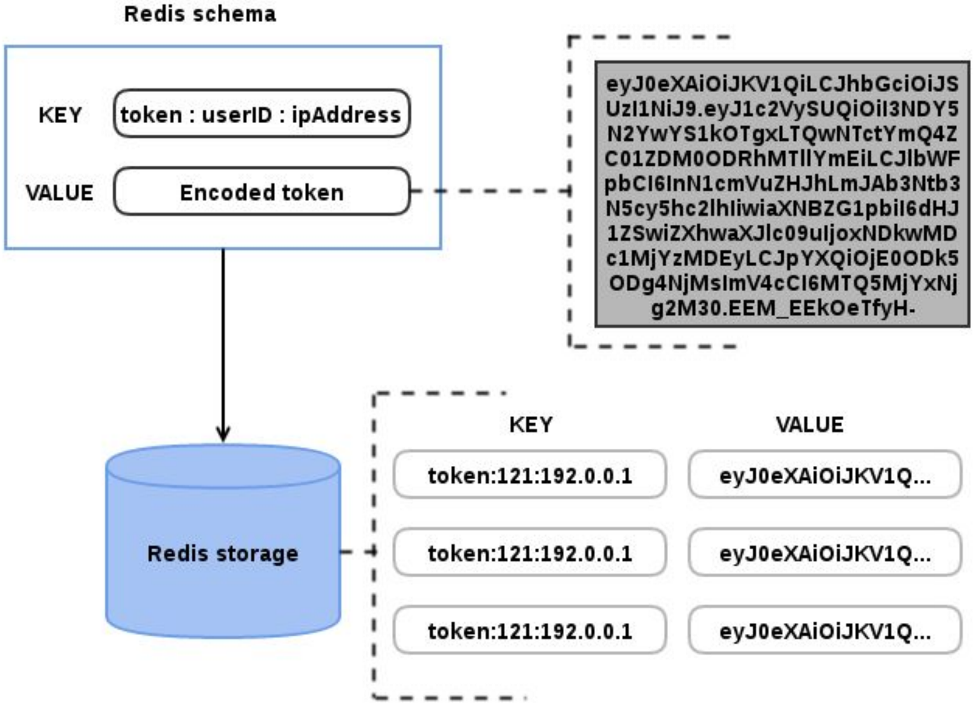
key	Value

User_id: Yang	Jenny: {
	unread_msg: 10;
	Last_msg_preview: happy_hour_at_10:..
	Timestamp: 21324
	}

	David: {
	unread_msg: 0;
	Last_msg_preview: what’s up?
	Timestamp: 341234;
	}

	Uncle: {
	unread_msg: 0;
	Last_msg_preview: dinner time?
	Timestamp: 1234; // expired
	}

Example 2



Redis schema

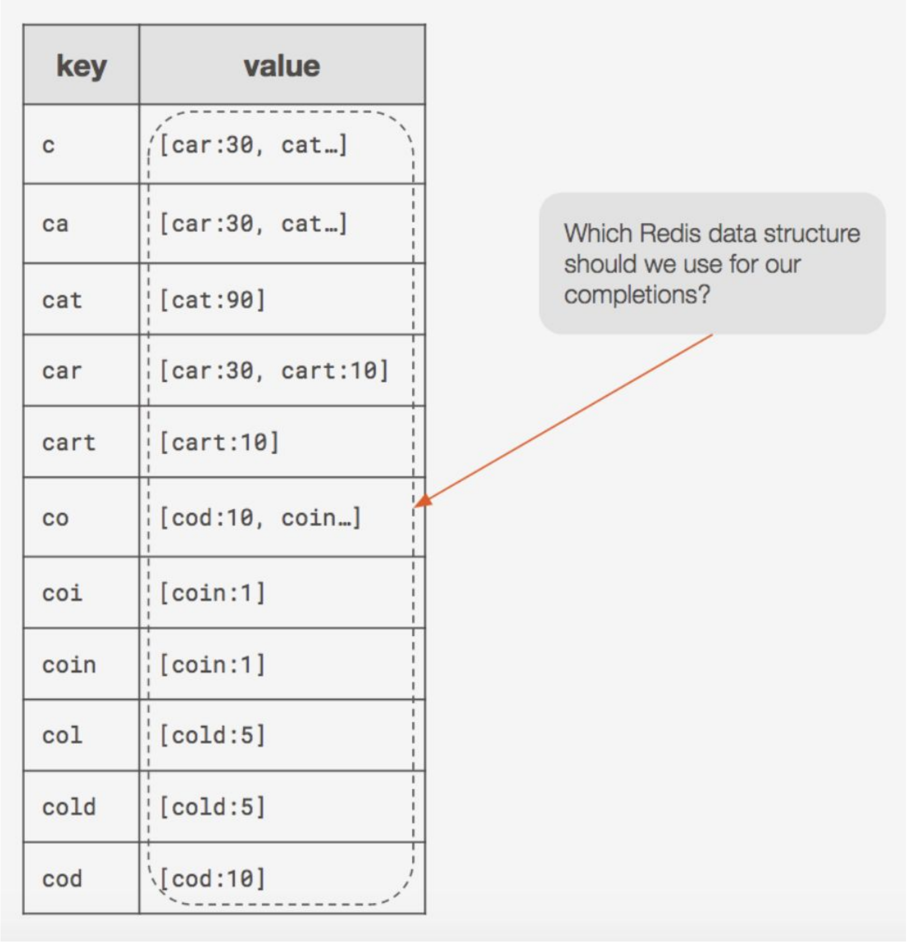
KEY	token : userID : ipAddress
VALUE	Encoded token

```
eyJ0eXAiOiJKV1QiLCJhbGciOiJS
UzI1NiJ9.eyJ1c2VySUQiOiI3NDY5
N2YwYS1kOTgxLTQwNTctYmQ4Z
C01ZDM0ODRhMTllYmEiLCJlbWF
pbCI6InN1cmVuZHJhLmJAb3Ntb3
N5cy5hc2lhIiwiaXNBZG1pbiI6dHJ
1ZSwiZXhwaXJlc09uIjoxNDkwMD
c1MjYzMDEyLCJpYXQiOjE0ODk5
ODg4NjMsImV4cCI6MTQ5MjYxNj
g2M30.EEM_EEkOeTfyH-
```

Redis storage

KEY	VALUE
token:121:192.0.0.1	eyJ0eXAiOiJKV1Q...
token:121:192.0.0.1	eyJ0eXAiOiJKV1Q...
token:121:192.0.0.1	eyJ0eXAiOiJKV1Q...

Example 3



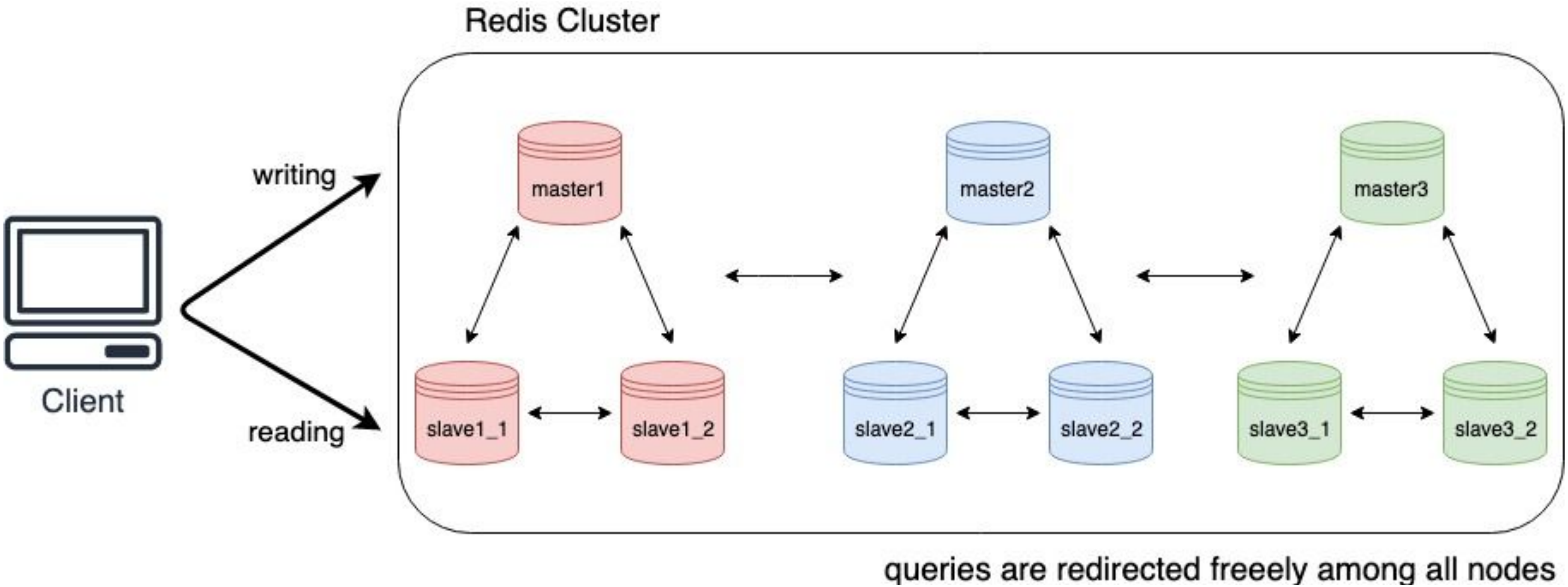
key	value
c	[car:30, cat…]
ca	[car:30, cat…]
cat	[cat:90]
car	[car:30, cart:10]
cart	[cart:10]
co	[cod:10, coin…]
coi	[coin:1]
coin	[coin:1]
col	[cold:5]
cold	[cold:5]
cod	[cod:10]

Which Redis data structure should we use for our completions?

Redis Persistence

- **RDB** (a.k.a. Redis Database Backup file): performs point-in-time **snapshots** of your dataset at specified intervals.
- **AOF** (a.k.a Append Only File): logs every write operation received by the server, that will be **replayed** again at server startup, reconstructing the original dataset.
- RDB vs. AOF
  - Document: https://redis.io/topics/persistence#rdb-advantages
  - Usually needs both

Scalability + High Availability => Redis Cluster



Redis Cluster

Consistent Hashing with Virtual Node + Master Slave Replication: quote from official website

Redis Cluster provides a way to run a Redis installation where data is ***automatically shared across multiple Redis nodes****.*

Redis Cluster also provides ***some degree of availability during partitions****, that is in practical terms the ability to continue the operations when some nodes fail or are not able to communicate. However the cluster stops to operate in the event of larger failures (for example when the majority of masters are unavailable).*

So in practical terms, what do you get with Redis Cluster?

- *The ability to* ***automatically split your dataset among multiple nodes****.*
- *The ability to* ***continue operations when a subset of the nodes are experiencing failures*** *or are unable to communicate with the rest of the cluster.*

….