Consistent Hashing

Shawn K.

Agenda:

- Background
- Data Partitioning
- Consistent Hashing to Rescue
- Virtual Nodes (Vnodes)
- Vnodes Advantages
- Data Replication (using Consistent Hashing)
- System Design Interviews
- Use Cases
- Demo w/ NGINX Load Balancer

Background

Important Aspect for Scalable System Design

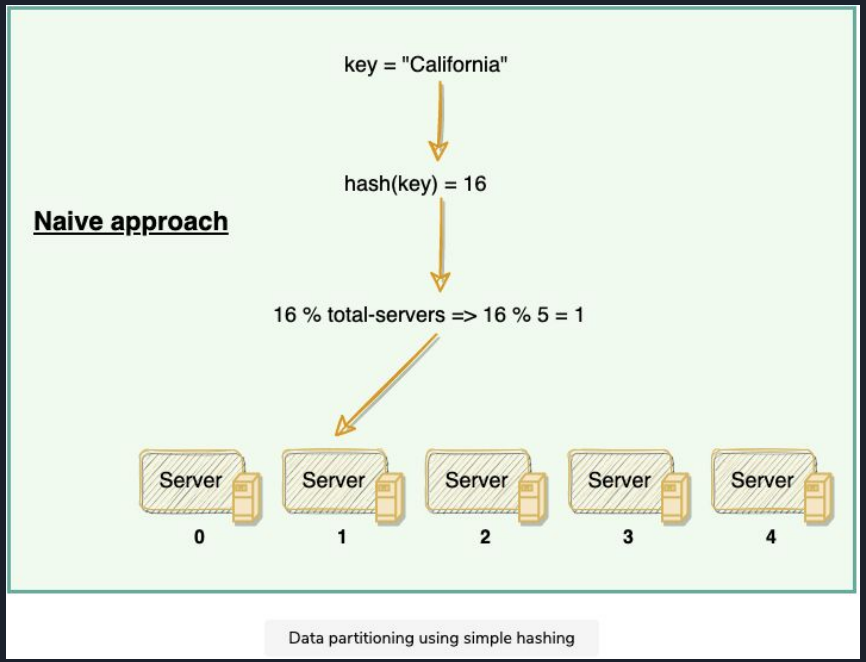
- **Data Partitioning**
  - Process of distributing data across servers
  - Improve scalability & performance
- **Data Replication**
  - Process of making X copies of data and storing them on different servers
  - Improve availability & durability of data across systems

What is data partitioning?

Act of distributing data across a set of nodes

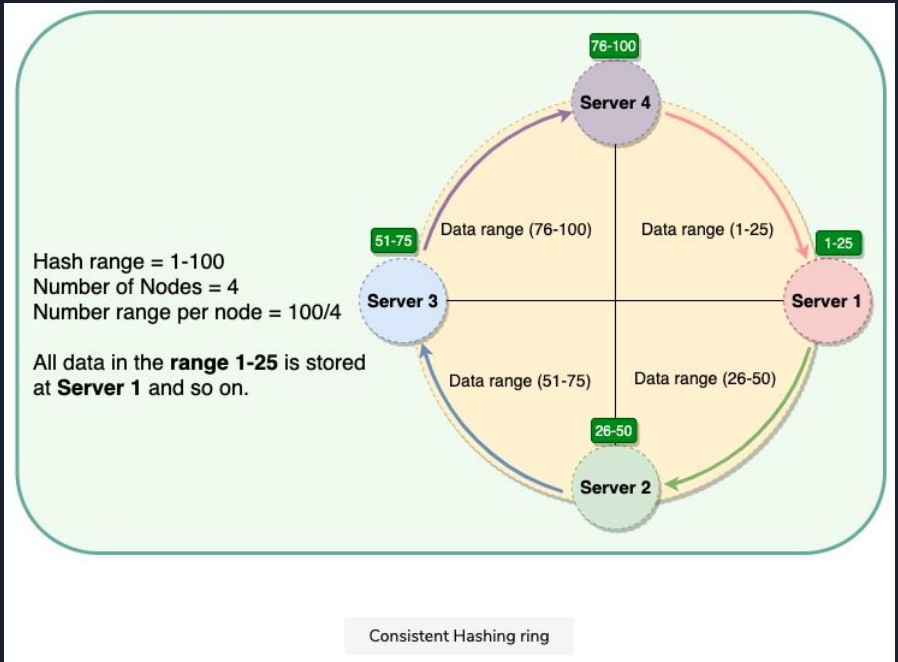
Challenges:

- How to know: Which node a particular piece of data will be stored?
- How to know: What data will be moved from existing nodes to the new nodes? (When adding or removing modes)
- How to minimize data movement when nodes join or leave?



Data partitioning using simple hashing

Consistent Hashing to the rescue (1/2)

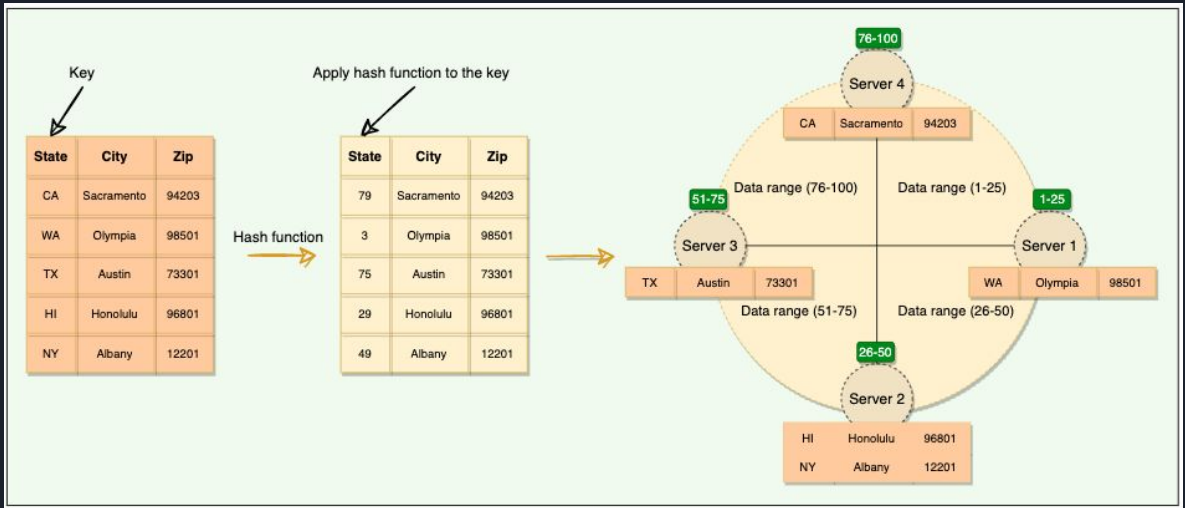


Consistent Hashing ring

Server	Token	Range Start	Range End
Server 1	1	1	25
Server 2	26	26	50
Server 3	51	51	75
Server 4	76	76	100

Consistent Hashing to the rescue (2/2)

- MD5 algo → key → store data for a fixed range
- Works well when adding/removing a node because next node takes the responsibility.
  - Non-uniform data and load distribution → virtual nodes will solve this problem.

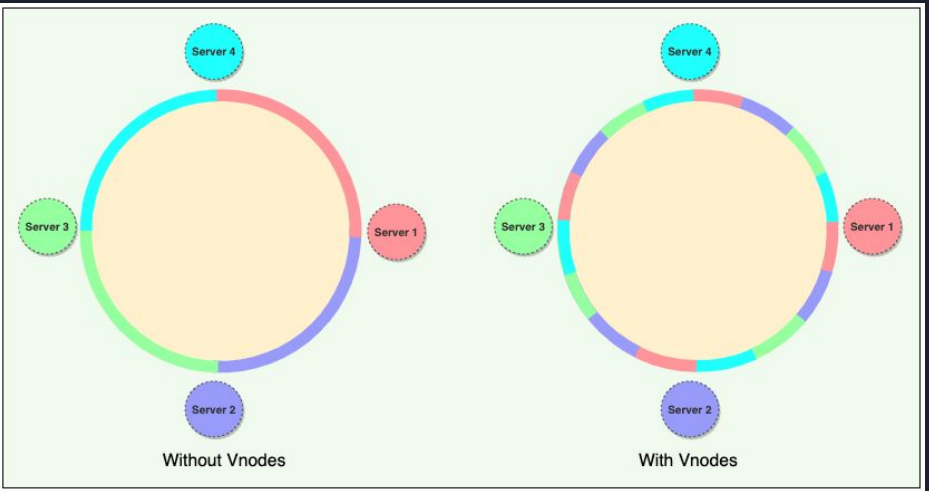


Distributing data on the Consistent Hashing ring

Virtual Nodes

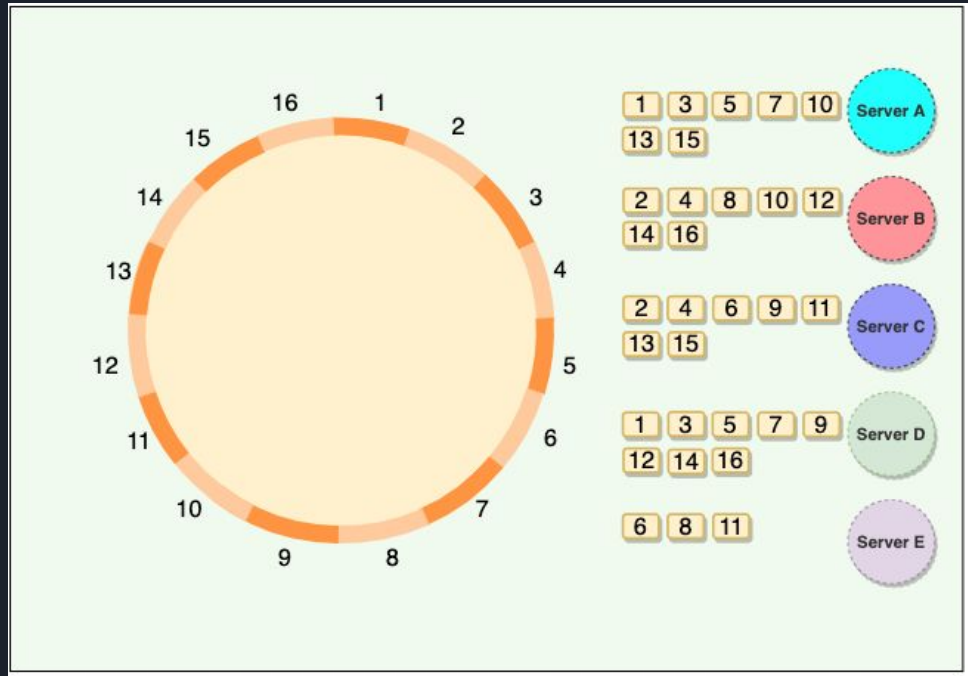
Basic Algorithm: a token(or hash range) → physical node

- **Adding/removing** nodes: recomputing→ overhead for large cluster
- **Hotspots**: if data isn't evenly distributed → some nodes (hotspot)
- **Node rebuilding**: replica nodes(pressure) → service degradation



Mapping Vnodes to physical nodes on a Consistent Hashing ring

New Scheme of distributing tokens to physical nodes



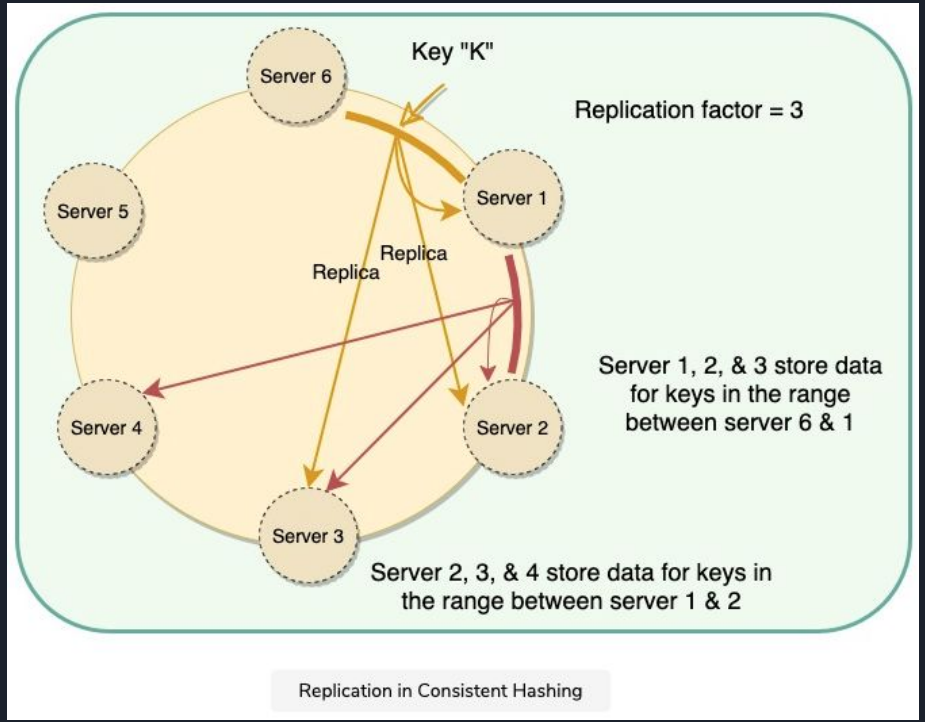
Mapping Vnodes to physical nodes on a Consistent Hashing ring

Advantage of Vnodes

- **Speeds up rebalancing process** (after adding/removing nodes)
  - Spread load more evenly across physical nodes by dividing hash ranges into smaller subranges
- **Easy to maintain** a cluster containing heterogeneous machines
  - High # of sub-ranges → powerful server
  - Low # of sub-ranges → less powerful server
- **Decrease hotspots probability**: assign smaller ranges to each physical node

Data Replication using Consistent Hashing

- Purpose: HA & Durability
- Replicate data → N nodes
  - N: replication factor
  - Asynchronous
  - Eventual consistency for HA



Replication in Consistent Hashing

Consistent Hashing in System Design Interviews

Helps w/ efficiently **partitioning & replicating** data

- **Scale up or down** storage/database servers based on usage (e.g., Christmas w/ high traffic)
- **Dynamic adjustment** of its **cache usage** (by adding/removing cache servers w/ traffic load)
- **Achieve HA**: replicate its data shards

Consistent Hashing Use Cases

Consistent Hashing Use Cases

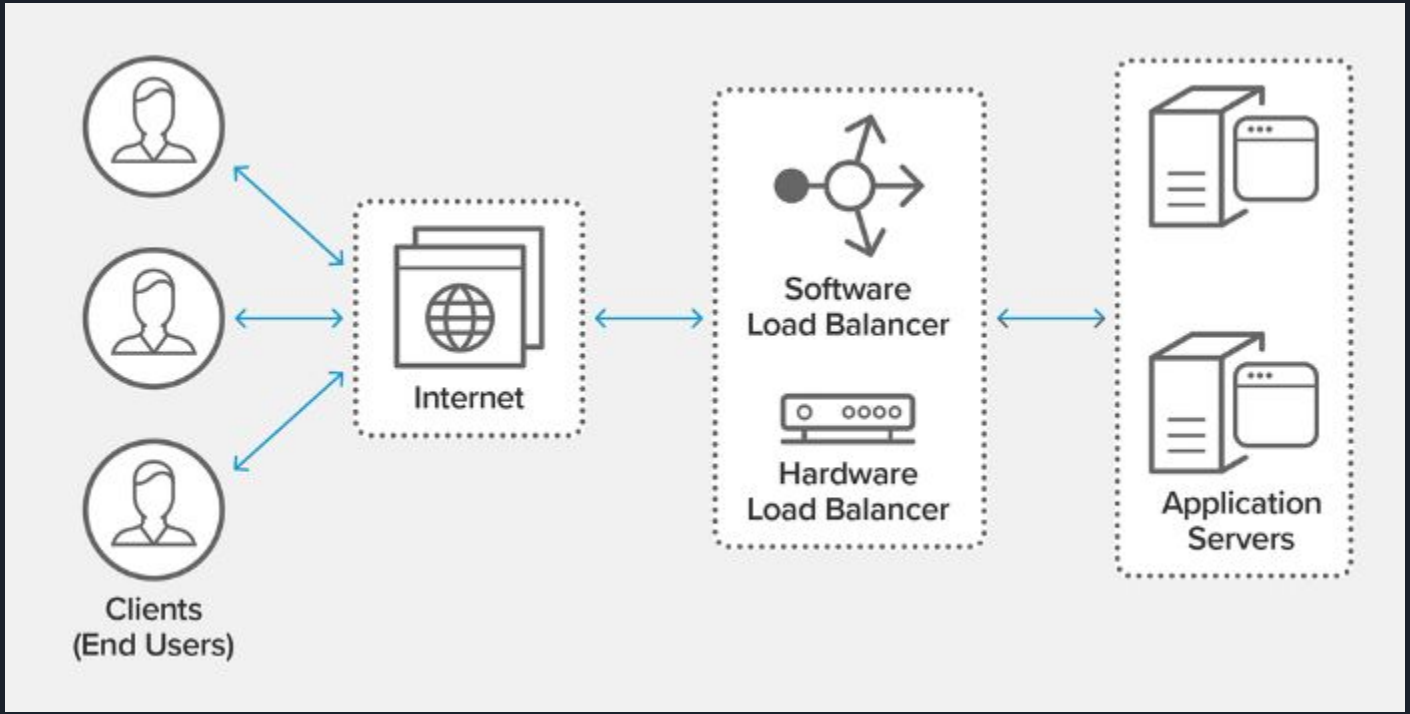
Distribute & replicate data across nodes:

- Amazon Dynamo
- Apache Cassandra

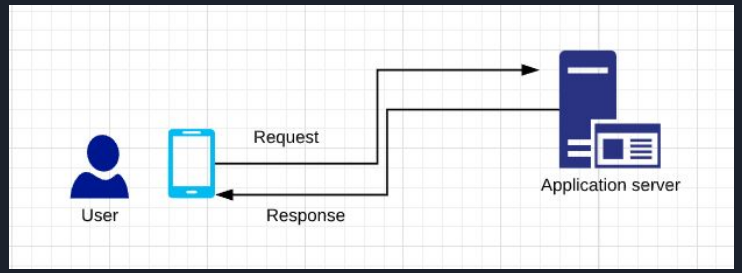
Load balancing large # of cache servers w/ dynamic content:

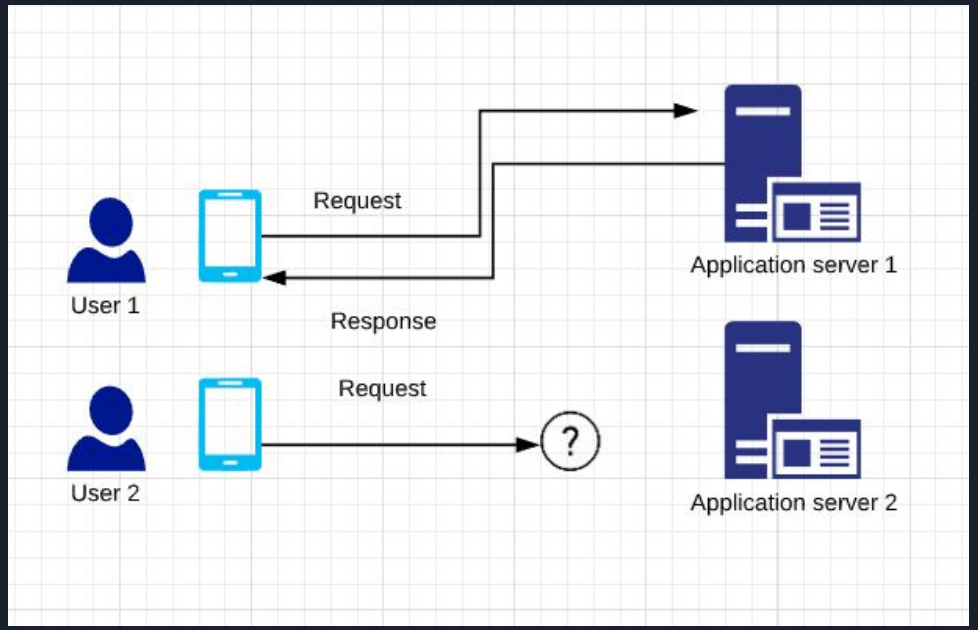
- Nginx Load Balancer
- Avi Load Balancer

Load balancing & Consistent Hashing



Load balancing & Consistent Hashing

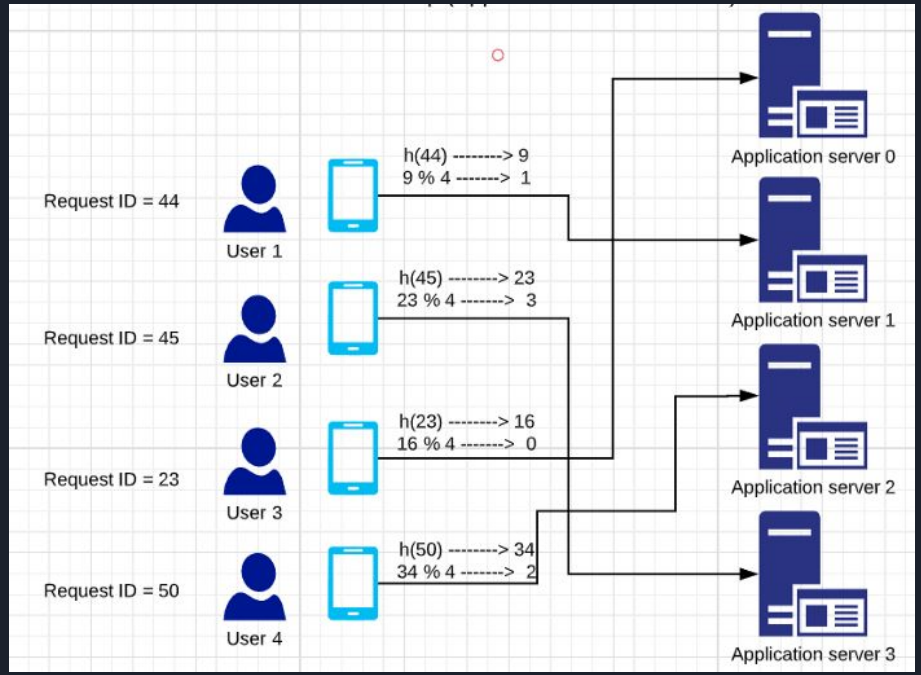




Load balancing & Consistent Hashing

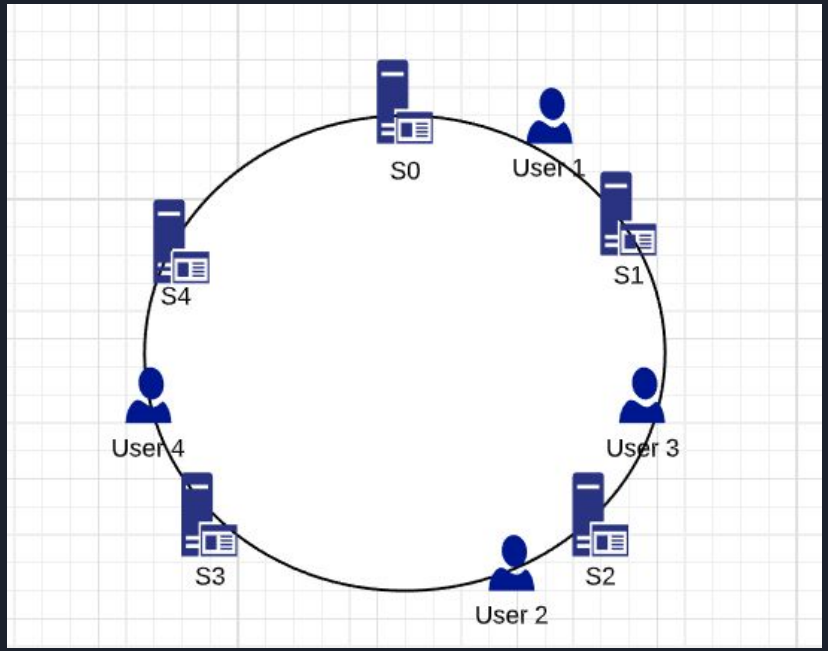
- Distribute weight using Hashing

r1 = Requests ID
n = Number of Servers
p = Application Server Number
h(r1) --------> m1
m1 % n --------> p (Application server Number)



Load balancing & Consistent Hashing

- Distribute weight using Hashing



Demo: NGINX Load Balancer w/ Consistent Hashing

https://github.com/changbal/nginx-consistent-hash

upstream load distribution by using an internal **consistent hash** ring to select backend nodes

```
Every 2.0s: docker ps --format "table {{.ID}}\t{{.Image}}\t{{.Ports}}\t{{.Names}}"

CONTAINER ID   IMAGE                                PORTS                                              NAMES
14f9d2d2edb6   nginx-load-balancer-consistent-hashing   0.0.0.0:9090->9090/tcp, 0.0.0.0:8080->80/tcp   nginx-load-balancer-consistent-hashing
22fd7e5c2f2e   video-service                        0.0.0.0:11001->80/tcp                              video-service-01
83a7f1649e54   video-service                        0.0.0.0:11002->80/tcp                              video-service-02
50e17e673bb3   video-service                        0.0.0.0:11003->80/tcp                              video-service-03
de2195aad71c   video-service                        0.0.0.0:11004->80/tcp                              video-service-04
197bb0f1f47d   video-service                        0.0.0.0:11005->80/tcp                              video-service-05
```