1. REST API Architecture style

XML {JSON} ... ...

```
{
  "username": "john_doe",
  "email": "john@example.com"
}
```


```
{
  "id": 123,
  "username": "john_doe",
  "email": "john@example.com"
}
```

1. **Web Applications**. Backend for web apps with CRUD operations.
2. **Mobile Apps**. Backend support for mobile app data needs.
3. **Public APIs**. Offer services to third-party developers.
4. **Integration**. Connect different systems or services.
5. **Content Delivery**. Distribute content to various platforms.
6. **Data Storage**. Backend for apps needing database access.
7. **E-Commerce**. Manage products, orders, and user accounts.
8. **Social Media Platforms**. Handle user data, posts, and interactions.
9. **IoT Devices**. Communicate with and manage IoT devices.
10. **Legacy System Interface**. Modern interface for older systems.

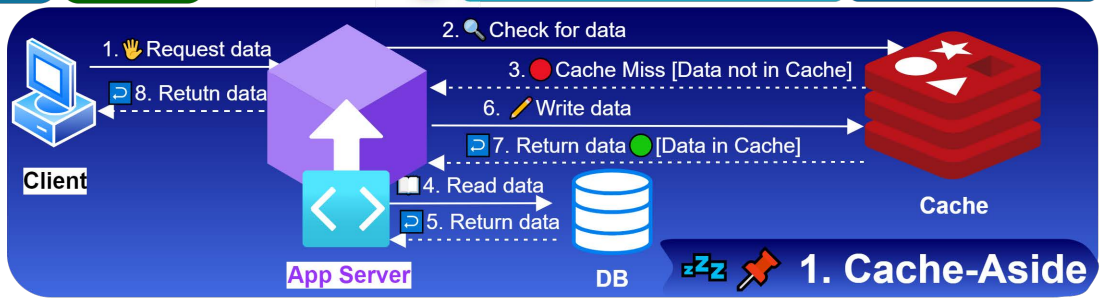
1. **Stateless**: Scales easily
2. **Cacheable**: Improved performance
3. **Uniform Interface**: Standardized use
4. **Scalable**: Supports high request volumes
5. **Interoperable**: Works well with HTTP
6. **Simple**: Uses standard HTTP methods
7. **Widely Adopted**: Many resources available

1. **Over/Under-fetching**: Fixed data returns
2. **Multiple Endpoints**: More requests for complex data
3. **Versioning Issues**: Changes can break clients
4. **Limited Flexibility**: Fixed data structures
5. **Performance**: Multiple trips for complex data.
6. **Stateful Limitations**: Not ideal for real-time operations
7. **Nested Resource Complexity**: Harder to manage nested data

Caching strategies Part 1

blog.ezvizi.com EZVIZI

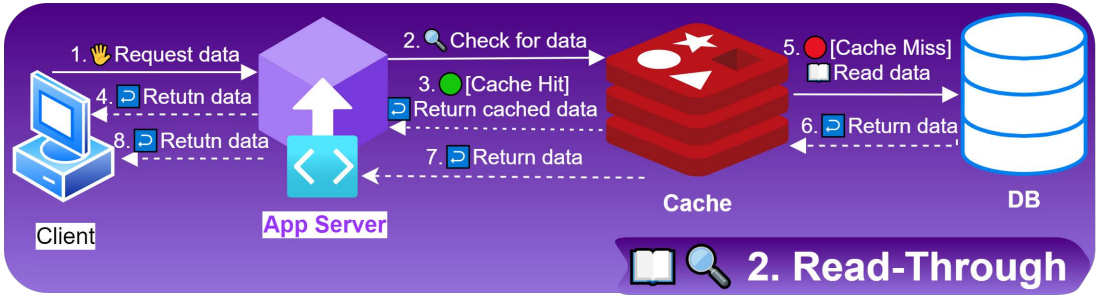
1. Cache-Aside



Pros
1. Efficient: Loads only needed data
2. Simple: App-driven cache management

Cons
1. Latency on cache miss.
2. App handles cache misses.
3. Risk of stale data.

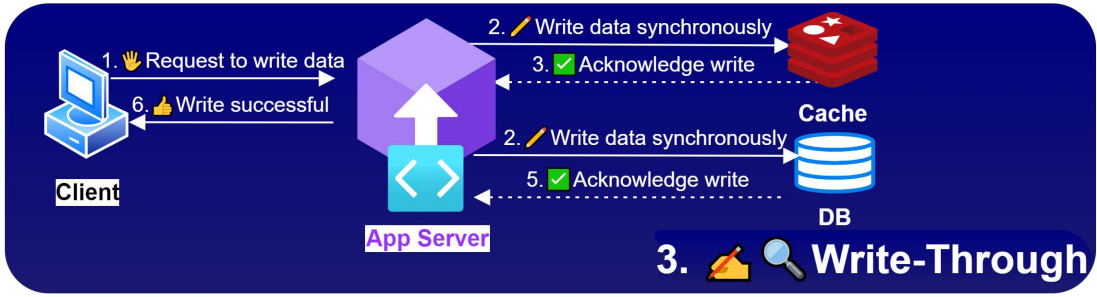
2. Read-Through



Pros
1. Reduced DB load
2. Always fresh data
3. Simplified Error Handling
4. Consistent response times
5. Automated data loading

Cons
1. Latency on cache miss
2. Complex cache management
3. Risk of stale data
4. Additional overhead
5. Resource consumption

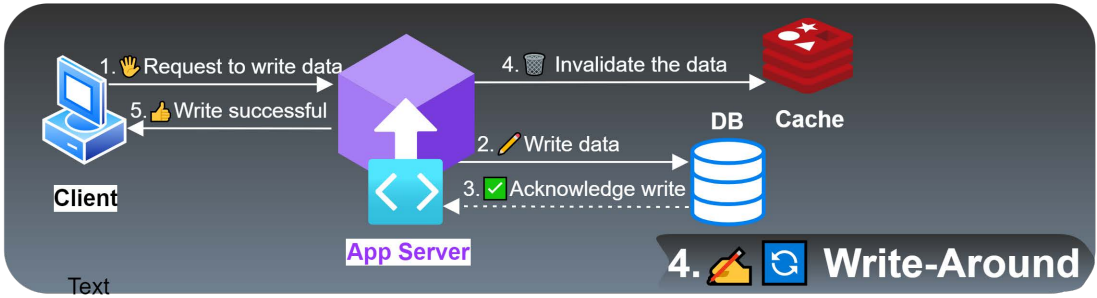
3. Write-Through



Pros
1. Ensures data consistency
2. Reduced risk of data loss
3. Immediate error detection
4. Fast subsequent reads

Cons
1. Slower write operations
2. Increased storage load
3. Resource overhead
4. Potential cache churn

4. Write-Around

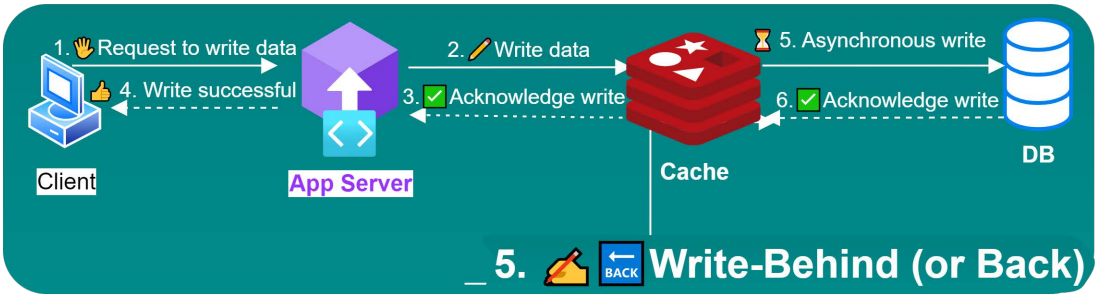


Text

Pros
1. Minimizes cache churn
2. Quick write operations
3. Efficient cache space usage
4. Good for read-intensive apps

Cons
1. Delay in initial read after write
2. Added management complexity
3. Risk of stale data

5. Write-Behind (or Back)



Pros
1. Fast write response
2. Batched Transfers
3. Lower Storage Stress

Cons
1. Risk of data loss
2. Temporary data mismatch
3. Added sync complexity

1.0.0 29 Sep 2023 EZP-001-2

Differences: API Gateway and Load Balancer

Key Differences:

Focus:

- **API Gateway: Focuses on API request management and processing.**
- **Load Balancer: Focuses on evenly distributing traffic.**

Granularity:

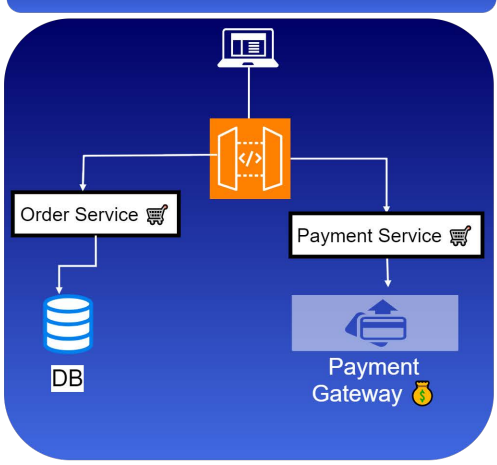
- **API Gateway: Offers fine-grained control over API requests.**
- **Load Balancer : Operates at a broader network level.**

Features:

- **API Gateway: Comes with a wide range of features tailored for API management.**
- **Load Balancer: Primarily deals with traffic distribution and server health.**

1. API Gateway

Primary Role: Manages and processes API requests



Use Cases:

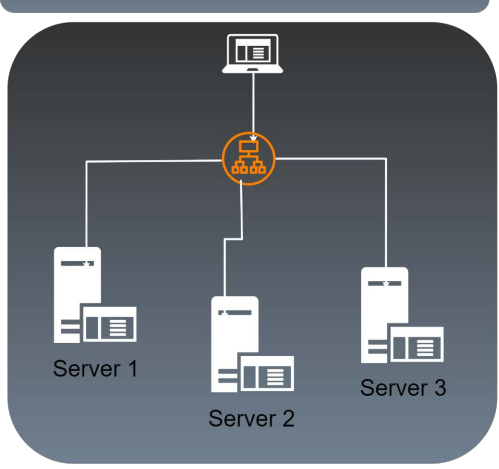
1. **Microservices architectures**
2. **API management**
3. **Aggregating responses from multiple services**
4. **Securing APIs**

Functions:

1. **Request Routing** : Directs requests to the appropriate service based on the API endpoint.
2. **API Composition** : Aggregates data from multiple services into a single response.
3. **Rate Limiting** : Limits the number of requests a user or system can make within a specified time.
4. **Security** : Provides features like authentication, authorization, and threat detection.
5. **Caching** : Stores frequently used data to speed up subsequent requests.
6. **Request/Response Transformation** : Modifies request or response format as needed.
7. **Analytics & Monitoring** : Tracks API usage and performance.
8. **Service Discovery** : Automatically detects and connects to available services.
9. **Error Handling & Retry** : Manages failed requests and retries if necessary.

2. Load Balancer

Primary Role: Distributes incoming network traffic across multiple servers



Use Cases:

1. **Ensuring high availability**
2. **Fault tolerance**
3. **Scaling applications**
4. **Managing traffic for large-scale web applications**

Functions:

1. **Traffic Distribution** : Evenly distributes incoming traffic to prevent server overload.
2. **Health Checks** : Monitors the health of servers and routes traffic away from unhealthy ones.
3. **SSL Termination** : Handles SSL/TLS decryption, offloading the task from servers.
4. **Session Persistence** : Ensures a user's session remains on the same server.
5. **Layer 4 and Layer 7 Load Balancing** : Can operate at both the transport (TCP/UDP) and application (HTTP/HTTPS) layers.

Linux Commands Part 1

blog.ezvizi.com

EZVIZI

1. File Operations

1. `ls` : List directory contents
   - Example: `ls -l`
2. `cp` : Copy files and directories
   - Example: `cp source.txt destination.txt`
3. `mv` : Move or rename files and directories
   - Example: `mv oldname.txt newname.txt`
4. `rm` : Remove files or directories
   - Example: `rm unwanted.txt`
5. `touch` : Create an empty file
   - Example: `touch newfile.txt`
6. `cat` : Concatenate and display file content
   - Example: `cat file.txt`

2. Directory Operations

1. `pwd` : Print working directory
   - Example: `pwd`
2. `cd` : Change directory
   - Example: `cd /home/user/documents`
3. `mkdir` : Make directories
   - Example: `mkdir new_directory`
4. `rmdir` : Remove empty directories
   - Example: `rmdir empty_directory`

3. System Info i

1. `uname` : Display system information
   - Example: `uname -a`
2. `top` : Display system tasks
   - Example: `top`
3. `df` : Disk space usage of file system
   - Example: `df -h`
4. `free` : Display memory usage
   - Example: `free -m`
5. `ps` : Display process status
   - Example: `ps aux`

4. Networking

1. `ping` : Send ICMP ECHO_REQUEST to network hosts
   - Example: `ping google.com`
2. `netstat` : Network statistics
   - Example: `netstat -tuln`
3. `ifconfig` : Display or configure a network interface
   - Example: `ifconfig eth0`
4. `ssh` : Secure shell client (remote login program)
   - Example: `ssh user@hostname`
5. `scp` : Secure copy (remote file copy program)
   - Example: `scp file.txt user@hostname:/path/`

5. Compression/Archiving

1. `tar` : Tape archiver
   - Example: `tar -czvf archive.tar.gz folder/`
2. `gzip` : Compress or expand files
   - Example: `gzip file.txt`
3. `gunzip` : Decompress files
   - Example: `gunzip file.txt.gz`
4. `zip` : Package and compress files
   - Example: `zip archive.zip file1.txt file2.txt`
5. `unzip` : Extract compressed files in a ZIP archive
   - Example: `unzip archive.zip`

6. Package Management

1. `apt-get` : APT package handling utility (Debian-based systems)
   - Example: `sudo apt-get install package_name`
2. `yum` : Package manager for RPM-based systems
   - Example: `sudo yum install package_name`
3. `dpkg` : Package manager for Debian
   - Example: `sudo dpkg -i package.deb`

7. Text Processing

1. `grep` : Search text
   - Example: `grep "pattern" file.txt`
2. `sed` : Stream editor
   - Example: `sed 's/old/new/g' file.txt`
3. `awk` : Pattern scanning and text processing language
   - Example: `awk '{print $1}' file.txt`

8. Help & Output

1. `man` : Display manual pages
   - Example: `man ls`
2. `echo` : Display a line of text
   - Example: `echo "Hello, World!"`

9. Process Management

1. `kill` : Terminate processes
   - Example: `kill -9 12345` (where 12345 is a process ID)

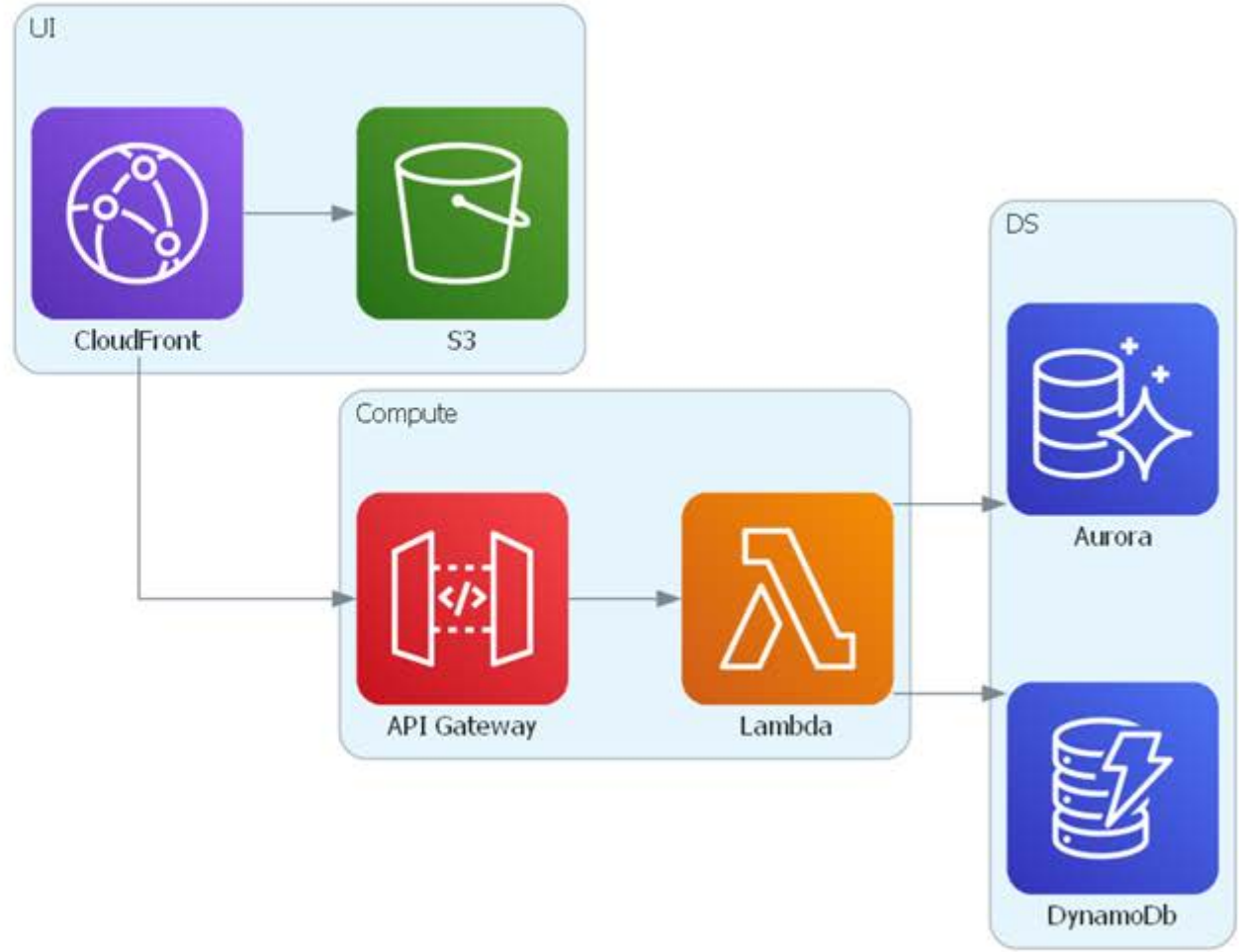
10.Permissions

1. `chmod` : Change file mode bits
   - Example: `chmod 755 script.sh`
2. `chown` : Change file owner and group
   - Example: `chown user:group file.txt`
3. `chgrp` : Change group ownership
   - Example: `chgrp group file.txt`

1.0.0 29 Sep 2023 EZP-001-4

diagram.py

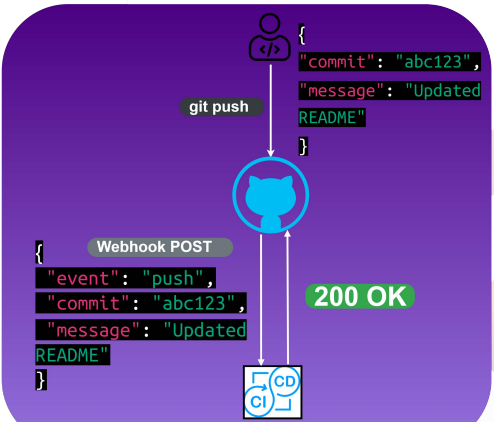
```python
# CF, S3, API GW, L, A, D
from diagrams import Diagram, Cluster
from diagrams.aws.network import CF
from diagrams.aws.storage import S3
from diagrams.aws.mobile import APIGateway
from diagrams.aws.compute import Lambda
from diagrams.aws.database import Aurora
from diagrams.aws.database import Dynamodb

with Diagram("Microservice", show = True):
    with Cluster("UI"):
        cf = CF("CloudFront")
        s3 = S3("S3")
    with Cluster("Compute"):
        api = APIGateway("API Gateway")
        l = Lambda("Lambda")
    with Cluster("DS"):
        a = Aurora("Aurora")
        d = Dynamodb("DynamoDb")
        cf >> s3
        cf >> api
        api >> l
        l >> a
        l >> d
```

Microservice

2. Webhooks API Architecture style

{JSON}



1. **Real-Time Notifications**. Immediate alerts for events like new posts.
2. **CI/CD**. Trigger builds, tests, and deployments on code changes.
3. **E-Commerce Updates**. Notify on order placements, payments, or stock changes.
4. **CMS**. Alerts on content changes.
5. **Social Media Integration**. Updates on posts and interactions.
6. **Monitoring & Analytics**. Alerts for system or security issues.
7. **Chatbots**. Notifications for new messages.
8. **Payments**. Status updates on transactions.
9. **IoT Device Communication**. Send data when specific conditions are met by IoT devices.
10. **Collaboration Tools Integration**. Notify on task updates or new messages in collaboration apps.

1. **Real-Time Notifications**: Immediate alerts without polling
2. **Event-Driven**: Supports dynamic, event-based interactions.
3. **Reduced Server Load**: Sends data only on events.
4. **Configurable**: Tailored to specific events
5. **Simple to Use**: Generally straightforward implementation
6. **Asynchronous Operations**: Non-blocking data transfers.
7. **Scalable**: Adapts easily with growing needs.

1. **Security Concerns**: Public endpoints may pose risks if not secured
2. **Error Handling**: Challenges with endpoint failures or errors
3. **Latency Issues**: Network delays can affect real-time nature
4. **Resource Use**: Each call consumes server resources
5. **Debugging**: Complexity increases with third-party services
6. **Limited Filtering**: Some providers may lack detailed options
7. **Ordering**: No guaranteed sequence for events.

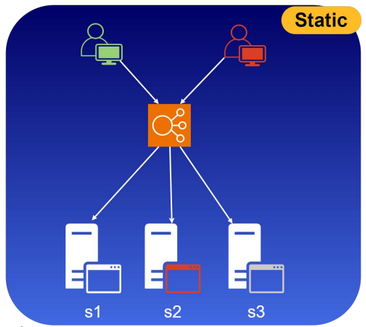
Load Balancing Methods Part 1

blog.ezvizi.com

EZVIZI

1. Round Robin

Static



N

```
upstream backend {
  server s1.dmn.com;
  server s2.dmn.com;
  server s3.dmn.com;
}
```

\+

1. Easy Setup 🟢: Minimal configuration
2. Even Distribution 🔁: Suitable for testing.
3. No Monitoring Needed 📊: Rotates through servers.
4. Predictable 🔄: Can anticipate server handling.

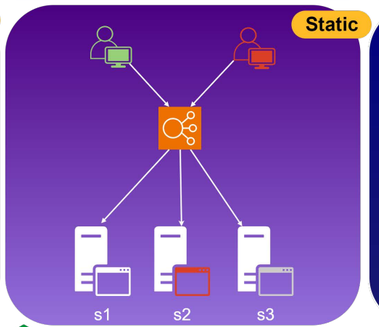
−

1. Server Overload Risk ⚠️: Can push servers to overload if they're already heavily loaded.
2. Requires Similar Server Capacity 📏: Best when all servers have roughly the same capacity.
3. Content Uniformity Needed 📋: Requires all servers to host the same content.

1. Testing Environments 🧪: Balanced request distribution for testing.
2. **Stateless Applications** 🔄: For independent, session-less requests.
3. **Uniform Server Capacities** 📏: When all servers have similar resources.
4. **Microservices** 🚀: Even distribution across stateless services.

2. Sticky Round Robin

Static



N+

```
upstream backend {
server s1.dmn.com;
server s2.dmn.com;
server s3.dmn.com;

sticky cookie srv_id expires=2h
domain=.dmn.com path=/;}
```

\+

1. Session Persistence 📌: Maintains user session data.
2. **Better User Experience** 🚀: No session timeouts or data loss.
3. **Simpler App Design** 🛠️: Assumes user requests hit the same server.

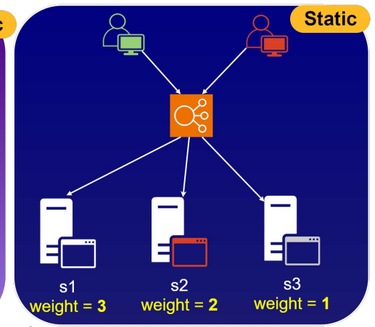
−

1. Potential Imbalance ⚖️: Uneven load distribution.
2. **Scaling Issues** 📈: Challenges when adding new servers.
3. **Server Failure Impact** 🔥: Disruptions if a sticky server fails.

1. Web Apps with Sessions 🛒: Maintains user carts and preferences.
2. Authentication Systems 🔒: Keeps users logged in across requests.
3. Multi-step Forms 📝: Remembers data across form pages.
4. Streaming Services 📹: Consistent server connection during streams.
5. Online Gaming 🎮: Tracks player states and scores consistently.

3. Weighted Round Robin

Static



N

```
upstream backend {
  server s1.dmn.com weight=3;
  server s2.dmn.com weight=2;
  server s3.dmn.com weight=1;}
```

\+

1. Adaptive Distribution 📊: Suits servers with different capacities.
2. **Flexibility** 🔄: Adjusts weights as server performance changes.
3. **Efficient** ⚙️: Maximizes resource utilization without overloading.

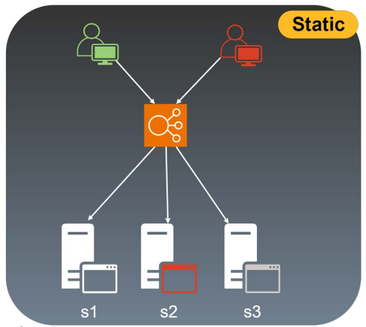
−

1. Complexity 🧩: Requires monitoring and weight adjustments.
2. **Potential Imbalance** ⚠️: Incorrect weights can lead to overloads.

1. Mixed Server Capacities 🎛️: When servers have varying resources.
2. Dynamic Environments 🌪️: Adapting to changing server performance.
3. Traffic Prioritization 🚦: Directing more traffic to higher-performing servers.

4. IP Hash

Static



N

```
upstream backend {
ip_hash;
  server s1.dmn.com;
  server s2.dmn.com;
  server s3.dmn.com;}
```

\+

1. Session Persistence 📌: Clients consistently directed to the same server.
2. **Predictable Distribution** 🔀: Based on client IP hash, ensuring even load.

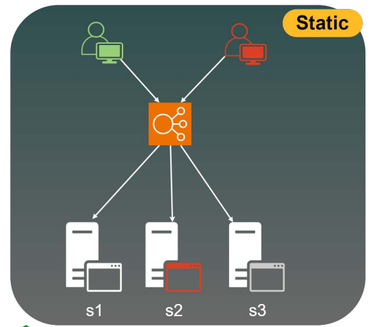
−

1. Limited Flexibility 🎛️: Hard to adjust once set up.
2. **Imbalance Risk** ⚠️: Some servers might get more traffic if certain IP ranges are more active.

1. Stateful Applications 🛒: Where session data needs to be retained.
2. **Geo-specific Content** 🌍: Serving content based on client's geographic location.
3. **Security & Monitoring** 🛡️: Easier tracking and management of client sessions.

5. Generic Hash

Static



N

```
upstream backend {
hash $request_uri;
  server s1.dmn.com;
  server s2.dmn.com;
  server s3.dmn.com;}
```

\+

1. **Versatility 🌐: Hashes on diverse inputs, including text, variables, or combinations like IP-port pairs or URIs.**
2. **Uniform Distribution 🔀: Aims for even distribution across servers.**

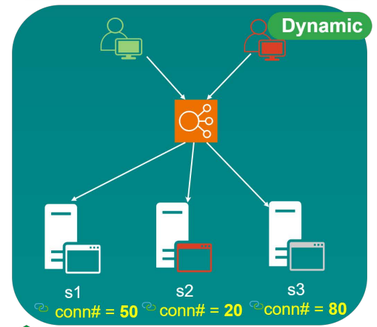
−

1. Complexity 🧩: Requires careful selection of hash function.
2. **Potential Imbalance** ⚠️: Hash collisions or poor hash functions can skew distribution.

1. Custom Inputs 🛠️: Hash based on specific application data or headers.
2. **Dynamic Environments** 🌪️: Where inputs for distribution change frequently.
3. **Cache Distribution** 💾: Ensuring cached content is evenly distributed.

6. Least Connections

Dynamic



N

```
upstream backend {
least_conn;
server s1.dmn.com;
server s2.dmn.com;
server s3.dmn.com;}
```

\+

1. Adaptive 🌐: Directs traffic to less-busy servers.
2. **Efficiency** ⚙️: Maximizes server utilization without overburdening.

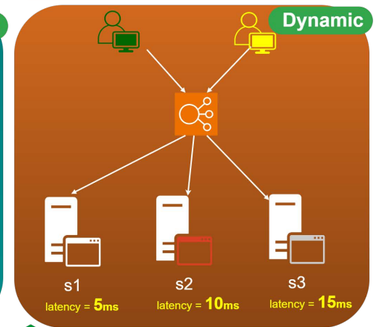
−

1. Delayed Reaction ⏱️: Might not account for sudden server load spikes.
2. **Potential Overhead** 📊: Requires monitoring of active connections.

1. Varying Server Capacities 🎛️: Balances load in mixed-capacity environments.
2. **High Traffic Sites** 🚦: Distributes large volumes of requests effectively.
3. **Real-time Applications** 🎮: Where quick response times are crucial.

7. Least Time

Dynamic



N+

```
upstream backend {
least_time header;
server s1.dmn.com;
server s2.dmn.com;
server s3.dmn.com;}
```

\+

1. Fast Responses 🚀: Prioritizes quickest servers.
2. **Dynamic Adaptation** 🌐: Adjusts based on server response times.

−

1. Monitoring Overhead 📊: Requires continuous tracking of server latencies.
2. **Fluctuation Risk** ⚠️: Rapid changes in response times can lead to frequent server switches.

1. User Experience 🎯: Ensures users get the fastest server response.
2. **High Demand Applications** 📹: For services like video streaming where latency matters.
3. **Variable Server Performance** 📉: Balances in environments with fluctuating server speeds.

Note: In NGINX, **`least_time`** combines the lowest number of active connections with the lowest average latency

1.0.0 6 Oct 2023 EZF-002-2

HTTP Status Codes

blog.ezvizi.com

EZVIZI

1xx: Informational

- 100 Continue: Initial request received; client can continue.
- 101 Switching Protocols: Server switching as client requested.
- 102 Processing: Request processing; no response yet.
- 103 Early Hints: Likely final response with header fields soon.

2xx: Successful

- 200 OK: Standard response for successful requests.
- 201 Created: Request fulfilled; new resource created.
- 202 Accepted: Request accepted but not yet completed.
- 203 Non-Authoritative Information: Response from another source.
- 204 No Content: Request processed; no content returned.
- 205 Reset Content: Request processed; client should reset view.
- 206 Partial Content: Partial resource delivered due to range header.
- 207 Multi-Status: Multiple status responses for WebDAV.
- 208 Already Reported: Enumeration has already been given (WebDAV).
- 226 IM Used: Request fulfilled with instance manipulations.

3xx: Redirection

- 300 Multiple Choices: Multiple options available.
- 301 Moved Permanently: Resource moved permanently to a new URL.
- 302 Found: Resource temporarily moved to a different URL.
- 303 See Other: Response found under another URL; use GET.
- 304 Not Modified: Resource unchanged since last request.
- 305 Use Proxy: Access resource via provided proxy.
- 307 Temporary Redirect: Temporary redirection to another URI.
- 308 Permanent Redirect: Permanent redirection to another URI.

5xx: Server Errors

- 500 Internal Server Error: Generic server error.
- 501 Not Implemented: Server can't fulfill request.
- 502 Bad Gateway: Invalid response from upstream server.
- 503 Service Unavailable: Server temporarily unavailable.
- 504 Gateway Timeout: Delayed response from another server.
- 505 HTTP Version Not Supported: Unsupported HTTP version.
- 506 Variant Also Negotiates: Configuration error in server.
- 507 Insufficient Storage: Can't store needed data. (WebDAV)
- 508 Loop Detected: Infinite loop detected. (WebDAV)
- 509 Bandwidth Limit Exceeded: Bandwidth limit surpassed.
- 510 Not Extended: Extensions needed for request.
- 511 Network Authentication Required: Client needs network authen.
- 520 Unknown Error: Unexpected server response.
- 521 Server Is Down: Origin server down.
- 522 Timeout: Server response delay.
- 523 Origin Unreachable: Can't reach origin server.
- 524 Timeout: Request delay.
- 525 SSL Handshake Failed: SSL handshake failed.
- 526 Invalid SSL Certificate: Invalid SSL certificate.
- 527 Railgun Listener to Origin: Railgun error.
- 529 Service Overloaded: Service is overloaded.
- 530 Site Frozen: Site administratively frozen.
- 561 Unauthorized: Unauthorized access.
- 598 Network Read Timeout: Network read timeout.
- 599 Network Connect Timeout: Client-side timeout.

4xx: Client Errors

- 400 Bad Request: Request can't be processed.
- 401 Unauthorized: Missing or invalid authentication.
- 402 Payment Required: Payment needed for request.
- 403 Forbidden: Server refuses to respond.
- 404 Not Found: Resource isn't available.
- 405 Method Not Allowed: HTTP method isn't supported.
- 406 Not Acceptable: Response can't match the list of acceptable values.
- 407 Proxy Auth Required: Client must authenticate with proxy.
- 408 Request Timeout: Server waited too long for request.
- 409 Conflict: Request conflicts with current state.
- 410 Gone: Resource was available but isn't now.
- 411 Length Required: Request needs Content-Length header.
- 412 Precondition Failed: Server doesn't meet request preconditions.
- 413 Payload Too Large: Request size exceeds limit.
- 414 URI Too Long: URI longer than server can interpret.
- 415 Unsupported Media Type: Media type isn't supported.
- 416 Range Not Satisfiable: Can't deliver requested range.
- 417 Expectation Failed: Server can't meet Expect header requirements.
- 418 I'm a Teapot: April Fools' joke; not used in real web communication.
- 419 Page Expired: Previously valid page has expired.
- 420 Method Failure/Enhance Your Calm:
  Method failure or rate limiting.
- 421 Misdirected Request:
  Request was directed at a server that can't produce a response.
- 422 Unprocessable Entity:
  Request structure is correct, but semantics are wrong. (WebDAV)
- 423 Locked: Resource that's being accessed is locked. (WebDAV)
- 424 Failed Dependency:
  Request failed due to a previous request's failure. (WebDAV)
- 425 Too Early: Server won't risk processing a possibly replayed request.
- 426 Upgrade Required: Client should switch protocol.
- 428 Precondition Required: Server requires request to be conditional.
- 429 Too Many Requests:
  Client sent too many requests in a given time frame.
- 430 HTTP Status Code: Unofficial status code.
- 431 Headers Too Large: Request headers size exceeds server limit.
- 440 Login Time-Out: Session has expired.
- 444 No Response: Server returns no info and closes connection.
- 449 Retry With: Request should be retried after performing an action.
- 450 Blocked by Parental Controls:
  Blocked by Windows Parental Controls.
- 451 Legal Reasons: Legal issues prevent resource availability.
- 460 Client Closed Connection Prematurely:
  Client closed connection before server response.
- 463 Too Many Forwarded IP Addresses:
  Too many IP addresses in the 'X-Forwarded-For' header.
- 464 Incompatible Protocol: Incompatible protocol version.
- 494 Request Header Too Large: Request header is too large.
- 495 SSL Certificate Error: Problem with the SSL certificate.
- 496 SSL Certificate Required: Client must provide an SSL certificate.
- 497 HTTP to HTTPS: Client sent an HTTP request to an HTTPS port.
- 498 Invalid Token: Invalid token provided.
- 499 Token Required/Client Closed:
  Token is required or client closed the connection.

1.0.0 9 Oct 2023 EZF-002-3

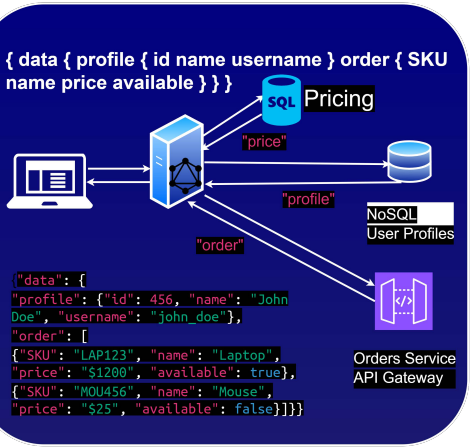
Fundamental Latency Metrics to Remember

Operation	Description	Time in ns	Time in ms	Software Example	Hardware/Networking Example
L1 cache reference	The time it takes to access data from the fastest, closest cache in a CPU.	1	-	Accessing a local variable in **GCC**-compiled C code.	Reading from L1 cache of **Intel Core i7**.
Branch misprediction	The penalty incurred when the CPU incorrectly predicts the next instruction to execute.	3	-	Mispredicted branch in a **Java** app due to JIT optimization.	Branch misprediction on **AMD Ryzen**.
L2 cache reference	The time to access the secondary cache, which is larger but slower than L1.	4	-	Function call overhead in **Swift**.	Accessing L2 cache of **Qualcomm Snapdragon**.
Mutex lock/unlock	The time to ensure exclusive access to a resource in concurrent programming.	17	-	Acquiring lock using **std::mutex** in C++.	Semaphore operation on **ARM Cortex-A**.
Main memory reference	The time to access data from the main system memory (RAM).	100	-	Accessing array in Go.	Fetching from Corsair Dominator DDR4 RAM.
Compress 1 kB with Zippy	The time to compress data for efficient storage or transmission.	2000	0.002	Compressing JSON with Google's Snappy.	Compression on NVIDIA Tegra chip.
Read 1 MB from memory	The time to read a large chunk of data from system memory.	10000	0.01	Buffering video in VLC Media Player.	Transfer in G.Skill Trident Z DDR4 RAM.
Send 2 kB over 10 Gbps	The time to send a small amount of data over a high-speed network.	1600	0.0016	Sending message using gRPC.	Transmitting over Cisco 10 Gbps switch.
SSD 4kB Random Read	The time to read a small, non-sequential chunk of data from an SSD.	20000	0.02	Loading config in Node.js.	Accessing Samsung 970 PRO NVMe SSD.
Read 1 MB from SSD	The time to read a large, sequential chunk of data from an SSD.	1000000	1	Loading assets in Unity.	Reading from Kingston A2000 NVMe SSD.
Round trip in datacenter	The time for a data packet to travel to a destination and back within the same data center.	500000	0.5	DB query in MongoDB on AWS.	Packet round trip on Juniper switch.
Read 1 MB from disk	The time to read a large, sequential chunk of data from a traditional spinning disk.	5000000	5	Loading image in Adobe Photoshop.	Reading from Seagate Barracuda HDD.
Read 1 MB from 1Gbps	The time to download a large chunk of data over a standard broadband connection.	10000000	10	Streaming segment in Netflix.	Downloading over Netgear 1 Gbps router.
Disk seek	The time for the read/write head of a HDD to move to the location of a specific piece of data.	10000000	10	Searching record in SQLite.	Seek on Western Digital Blue HDD.
TCP packet round trip	The time for a data packet to travel from one continent to another and back.	150000000	150	Accessing WordPress site in Europe from US.	Round trip over Verizon's international link.

Action & Description	Git Command
Git config: Set username and email.	`git config --global user.name "Alice Smith" git config --global user.email "alice.smith@example.com"`
Git init: Initialize a new repository.	**git init my-web-app**
Git clone: Clone a repository.	**git clone https://github.com/alice/my-web-app.git**
Git status: Check file status.	**git status**
Git add: Add file to staging.	**git add index.html**
Git commit: Commit changes.	**git commit -m "Added homepage."**
Git push: Push changes to remote.	**git push origin master**
Git branch: Create a new branch.	**git branch feature-navbar**
Git checkout: Switch to a branch.	**git checkout feature-navbar**
Git merge: Merge a branch.	**git merge feature-navbar**
Git pull: Pull latest changes.	**git pull origin master**
Git log: View commit history.	**git log**
Git show: Check last commit details.	**git show**
Git diff: Check differences.	**git diff**
Git tag: Tag a commit.	**git tag v1.0**
Git rm: Remove a file.	**git rm old-file.txt**
Git stash: Temporarily save changes.	**git stash**
Git reset: Undo last commit.	**git reset HEAD~1**
Git revert: Undo a specific commit.	**git revert commit_id**
Git remote: Check remote repositories.	**git remote -v**
Git fetch: Fetch changes without merging.	**git fetch origin**

3. GraphQL API Architecture style

{JSON}



1. **Dynamic Data Retrieval**. Fetch varied data without backend changes.
2. **Mobile Applications**. Optimize data for bandwidth & reduce requests.
3. **Single Page Applications (SPAs)**. Real-time data without page reloads.
4. **Data Source Integration**. Aggregate data from multiple sources.
5. **Social Media Platforms** .Manage nested comments, posts, profiles.
6. **Content Management (CMS)**. Serve flexible content structures.
7. **E-Commerce**. Fetch product data, reviews, profiles.
8. **Real-time Apps**. Support chat apps or live score updates.
9. **Development Tooling**. Benefit from API introspection.
10. **Decoupled Architectures**. Frontend requests data without backend changes.

1. **Flexible Queries**: Request specific data
2. **Single Endpoint**: Unified access point
3. **Strongly Typed**: Predictable results
4. **Introspection**: Discoverable schema
5. **Real-time Data**: Supports subscriptions
6. **Evolution Without Versioning**
7. **Batching & Caching**: Efficient data retrieval
8. **Unified Data Retrieval**

1. **Complexity**: Can be overkill for simple APIs.
2. **Performance Concerns**: Potential for resource-intensive queries
3. **Caching Challenges**: Traditional HTTP caching might not work
4. **Rate Limiting**: Granular rate limiting needed
5. **Learning Curve**: Different from REST
6. **Overhead**: Potential overhead for simple queries
7. **File Uploads**: No native support
8. **Error Handling**: Always returns 200 OK status.

NETFLIX

What It's Built On

Frontend

- Web
  - node
  - HTML
  - JS
  - React
- Mobile
  - Android: Kotlin
  - iOS: Swift

blog.ezvizi.com EZVIZI

GraphQL

Backend

- Cloud Infrastructure: aws — majority of Netflix's cloud infrastructure
- Microservices Architecture
  - microservices
    - Q2 2023
      - ~2000+ developers
      - ~100+ teams
      - paid subscribers: ~238M
      - ~1000+ microservices
    - Development: spring boot, Java
  - API gateway: NETFLIX ZUUL
  - Service Discovery: NETFLIX EUREKA
- Data Storage & Databases
  - Relational databases: MySQL, CockroachDB
  - NoSQL databases: cassandra, Amazon DynamoDB
  - Content/Streaming: Amazon S3, Amazon Elastic Transcoder
  - Search engines: elasticsearch
  - Caching: EVCache
- Stream processing systems: kafka, Flink
- Batch processing systems: Spark
- Big Data & Analytics
  - Data warehousing solutions: amazon REDSHIFT, snowflake
  - Data Visualization: tableau
  - Real-time analytics: druid
  - Enhance data performance: ICEBERG
- Machine Learning: TensorFlow
- CDN
  - NETFLIX Open Connect
  - Amazon CloudFront
- Networking
  - Load Balancing: Amazon ELB, NETFLIX ZUUL
  - Service Mesh: envoy
- Scripting & Automation: python, Groovy
- Incident management: PagerDuty

DevOps

- CI/CD: Spinnaker, Jenkins
- Configuration Management: ARCHAIUS
- Monitoring & Observability
  - Time-series monitoring: Apache Atlas
  - Logging systems: logstash, elasticsearch, kibana
  - Tracing tools: ZIPKIN
- Resilience & Chaos Engineering
- Version Control: git, GitHub
- Infrastructure as Code: Terraform
- Collaboration & Communication: slack, Jira, Confluence
- Build Automation & Dependency Management: Gradle
- Enhance the build and release process: nebula

Sources

- https://netflixtechblog.com/
- https://aws.amazon.com/solutions/case-studies/innovators/netflix/
- https://github.com/Netflix
- 1.0.0 EZF-003-2 Sep 30 2023

Disclaimer: The Netflix tech stack listed is not complete; this is only the information that is known. It's not guaranteed that the data here is the most recent.

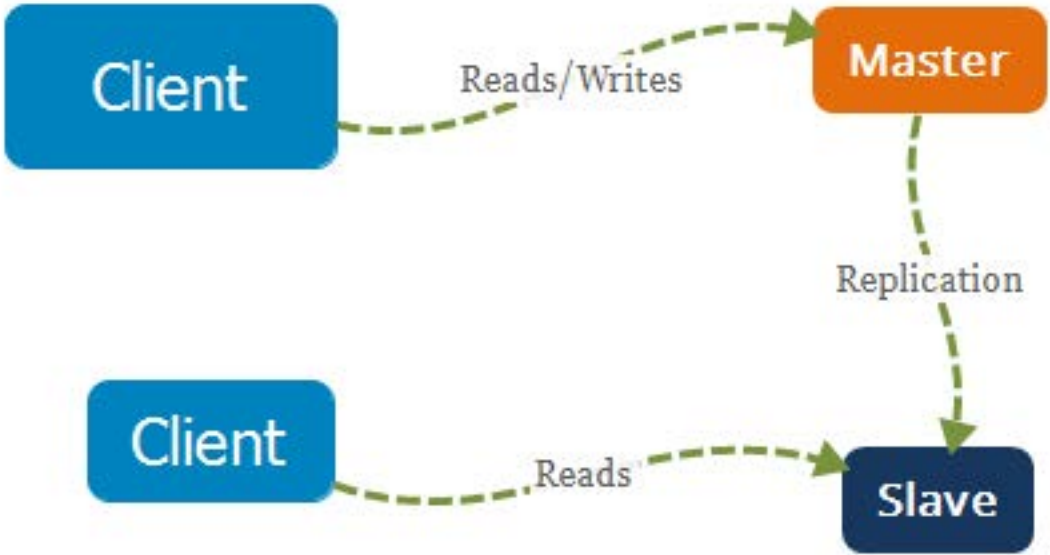
Comparison of URI, URL, and URN

- URI (Uniform Resource Identifier)
  - syntax: scheme:[//[user:password@]host[:port]][/]path[?query][#fragment]
    - Scheme/Protocol — Example: mailto
    - Authority
      - User Information (optional) — Example: ezvizi.com@gmail.com
      - Host/Domain Name
      - Port (optional)
    - Path: /products/page1
    - Query (optional) — Example: subject=Ad Placement Request
    - Fragment/Anchor (optional)
    - Example: **mailto:ezvizi.com@gmail.com?subject=Ad Placement Request?&cc=ezvizi.com@gmail.com&bcc=ezvizi.com@gmail.com**
  - Purpose: **To identify a resource**
  - Persistence: Can be either persistent or transient
  - Definition: A generic term for any type of name or address referring to a resource.
- blog.ezvizi.com
- URL (Uniform Resource Locator)
  - syntax: scheme:[//[user:password@]host[:port]][/]path[?query][#fragment]
    - Scheme/Protocol — Example: https
    - Authority
      - User Information (optional) — Example: user:password@
      - Host/Domain Name — Example: www.ezvizi.com
      - Port (optional) — Example: 8080
    - Path — Example: /products/page1
    - Query (optional) — Example: ?item=123
    - Fragment/Anchor (optional) — Example: #section2
  - Example: **https://user:password@www.ezvizi.com:8080/products/page1?item=123#section2**
  - Definition: A specific type of URI that describes where a resource is located and how to access it.
  - Purpose: **To locate a resource**
  - Persistence: Typically transient; can change if the resource moves
- https://www.linkedin.com/in/ildarkhanov/
- URN (Uniform Resource Name)
  - syntax: urn:<namespace-identifier>:<namespace-specific-string> — Example: **urn:isbn:0451450523**
  - Definition: A specific type of URI that provides a persistent identifier for a resource without implying its location or how to access it.
  - Purpose: **To name a resource uniquely**
  - Persistence: Meant to be persistent; doesn't change even if the resource's location changes.
- 1.0.0 EZF-003-4 Oct 3 2023

27 Microservices Best Practices

1. Independent Deployment and Scalability
   - Deploy and scale each microservice separately
2. Domain-Driven Design (DDD)
   - Model services based on business domains to ensure alignment with business capabilities.
3. Loose Coupling & High Cohesion
   - Minimize inter-service dependencies and group related functionalities.
4. API Versioning & Backward Compatibility
   - Ensure older versions of services continue to function
     - URI versioning
     - header versioning
     - parameter versioning
5. Service Discovery
   - Tools
     - CONSUL
     - NETFLIX EUREKA
     - kubernetes
   - Kubernetes service discovery
   - Don't use hardcoded values
6. Centralized Configuration Management
   - Tools
     - Spring Cloud Config
     - CONSUL
7. Health Checks
   - health endpoints for monitoring service health
   - Tools
     - kubernetes
     - CONSUL
8. Distributed Tracing
   - Trace requests across services for diagnostics
   - Tools
     - JAEGER
     - ZIPKIN
9. Micro Frontends
   - Break monolithic frontend into smaller, more manageable pieces
10. Data Consistency
    - patterns
      - Saga
      - eventual consistency
11. Centralized Logging
    - ELK stack
      - elasticsearch
      - logstash
      - kibana
    - graylog
    - easier debugging
12. API Gateway
    - Route requests, handle authentication, single entry point
    - Kong
    - NETFLIX ZUUL
13. Authentication & Authorization
    - Authentication
      - JWT
      - OAuth 2.0
    - Authorization
      - service level
    - ensure only authentic users access services
14. Rate Limiting & Request Caching
    - Protect services from excessive requests
    - reduce load on services
15. Database Per Service
    - Dedicate a database to each service for decoupling and data consistency.
16. Event-Driven Architecture
    - Tools
      - kafka
      - RabbitMQ
    - Communicate asynchronously between services
17. Backup & Disaster Recovery
    - Regularly backup data and test disaster recovery processes
18. CI/CD & DevOps Practices
    - Automate
      - build
      - test
      - deployment processes
    - Each microservice should have its own build and deployment pipeline
    - comprehensive testing for each service
    - ensure development teams own and support changes from development to end-of-life
    - blog.ezvizi.com
    - https://www.linkedin.com/in/ildarkhanov/
19. Containerization
    - docker
20. Orchestration
    - kubernetes
21. Single Responsibility Principle
    - Each service should cater to one business capability
22. Statelessness
    - Design for scalability
23. Resilience & Fault Tolerance
    - Tools
      - HYSTRIX Defend Your App
      - Resilience4j
    - Prevent cascading failures
      - Circuit Breakers
      - Automatic Retries
    - Design services to handle failures gracefully
24. Security
    - authentication
    - data encryption
    - API
    - network security
25. Code Maturity
    - Keep code within a service at a similar level of maturity
26. Documentation
    - Swagger
    - Tools
    - Maintain up-to-date API documentation
27. Monitoring & Alerts
    - Prometheus
    - Grafana
    - Tools

1.0.0 EZF-003-3 Sep 30 2023


Client
Reads/Writes
Master
Replication
Client
Reads
Slave

blog.ezvizi.com

EZVIZI

Networking & Content Delivery: AWS, Azure, GCP, OCI

Estimated degree of Match (1..5)
1 = "Not a Direct Match, but Closest Available"
2 = "Somewhat Similar"
3 = "Moderately Similar"
4 = "Very Similar"
5 = "Almost Identical"

1.0.0 17 Oct 2023 EZP-004-3

https://www.linkedin.com/in/ildarkhanov/

Networking & Content DeliveryService	aws	Azure Degree of Match (1..5) to AWS		Google Cloud Platform Degree of Match (1..5) to AWS		ORACLE Cloud Infrastructure Degree of Match (1..5) to AWS	
1. API Management Service	Amazon API Gateway	Azure API Management	5	Cloud Endpoints	5	API Gateway	5
2. CDN	Amazon CloudFront	Azure Content Delivery Network (CDN)	5	Cloud CDN	5	Content and Experience Cloud	3
3. Domains and DNS	Amazon Route 53	Azure DNS	5	Cloud DNS	5	OCI DNS	5
4. Secure access to apps	AWS Verified Access	Azure Active Directory	3	Identity Platform	3	*No direct equivalent in OCI*	-
5. Virtual Networks	Amazon VPC	Azure Virtual Network	5	Virtual Private Cloud (VPC)	5	Virtual Cloud Network (VCN)	5
6. Application networking	Amazon VPC Lattice	Azure Virtual WAN	1	*No direct equivalent in GCP*	-	*No direct equivalent in OCI*	-
7. Service mesh	AWS App Mesh	Open Service Mesh on AKS	4	Traffic Director	4	*No direct equivalent in OCI*	-
8. Services discovery (DNS)	AWS Cloud Map	Hashicorp Consul Service on Azure	2	Service Directory	4	*No direct equivalent in OCI*	-
9. Network connectivity	AWS Direct Connect	Azure ExpressRoute	5	Cloud Interconnect	5	FastConnect	5
10. Premium networking	AWS Global Accelerator	Azure Front Door; Cross-region LB	3	Network Service Tiers, Premium Tier	3	*No direct equivalent in OCI*	-
11. Private link	AWS PrivateLink	Azure Private Link	5	Private Service Connect	3	Service Gateway	4
12. Private mobile network	AWS Private 5G	Azure Private 5G Core	3	*No direct equivalent in GCP*	-	*No direct equivalent in OCI*	-
13. Network connectivity	AWS Transit Gateway	Azure Virtual WAN	4	Network Connectivity Center	4	Transit Routing: VCN w/ Transit Gateway	4
14. Network connectivity	AWS VPN	Azure VPN Gateway	5	Cloud VPN	5	VPN Connect	5
15. Load balancing	Elastic Load Balancing	Azure Load Balancer	5	Cloud Load Balancing	5	Load Balancing	5