Cloud Design Patterns

Ahmed Hesham

Cache Aside

- **Store frequently accessed data into a cache that is separate from a data store to improve performance**

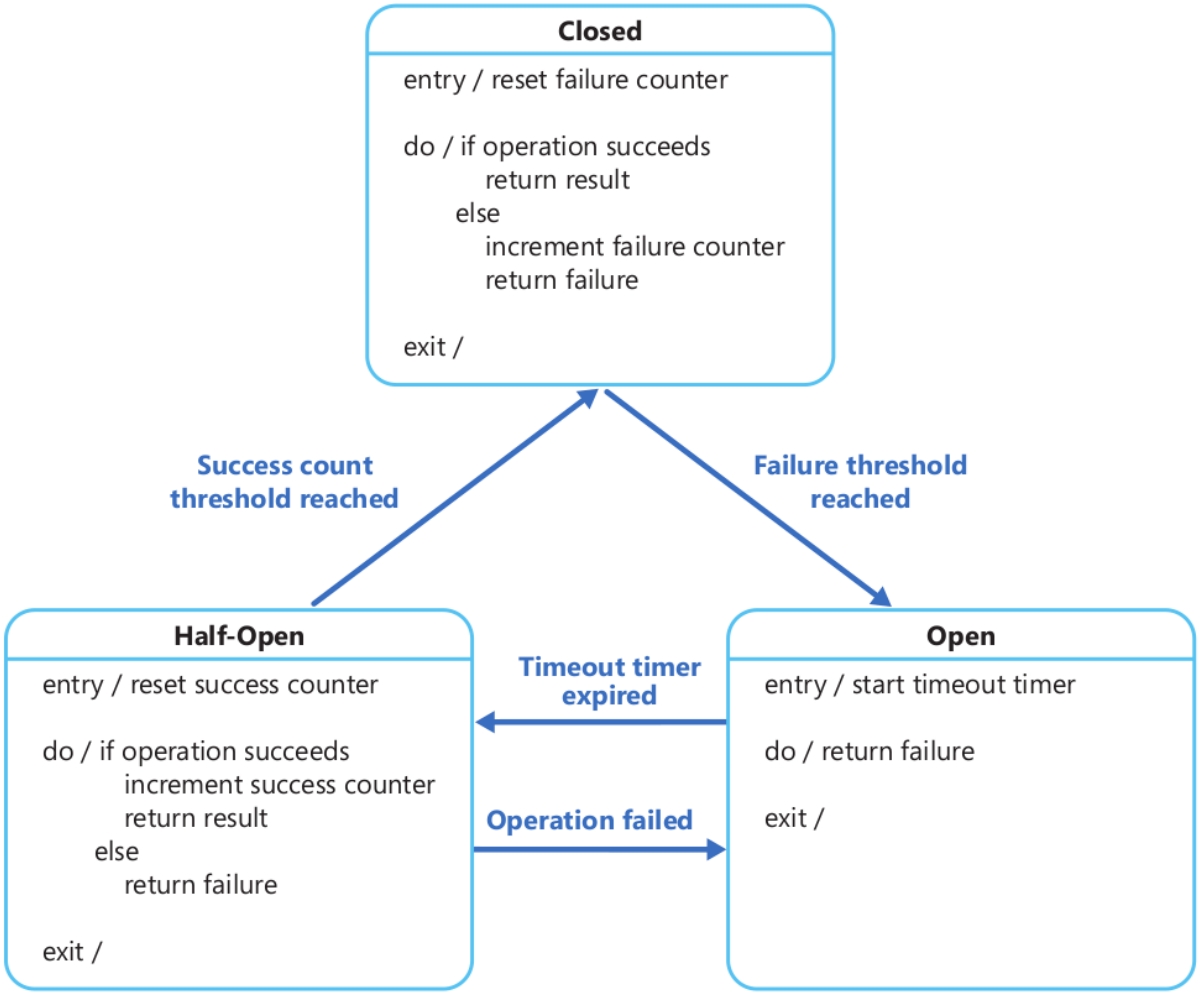
Issues To Consider

- Lifetime of cached item
- Cache eviction algorithm
- Cold boots for cache
- Consistency may not be guaranteed if a cache is external to the data store

Circuit Breaker

- Faults can occur at anytime in a cloud application
- It is important to not waste resources trying to perform actions that will fail
- For example trying to connect a remote server and the connection is down
- Circuit breaker acts as a Proxy between two entities

Circuit Breaker States



Issues To Consider

- Concurrency: the same circuit breaker would be accessed by many instances of an entity, it should not block concurrent requests
- Avoid using a single breaker on an entity that has providers, for instance in a distributed database, a node may be down but the rest are not

Issues To Consider – II

- **Accelerated Circuit Breaking: a response may contain enough information to trigger a break rather than wait for the counter**
- **Manual Override: it is best to implement a manual override for the admin to utilize, to force open or close the circuit**

Retry Pattern

- Similar to Circuit Breaker, however Circuit Breaker assumes that the request will fail and therefore will deny further requests for a period of time, while Retry pattern will assume that the request will succeed if it is sent again
- More suited for transient errors

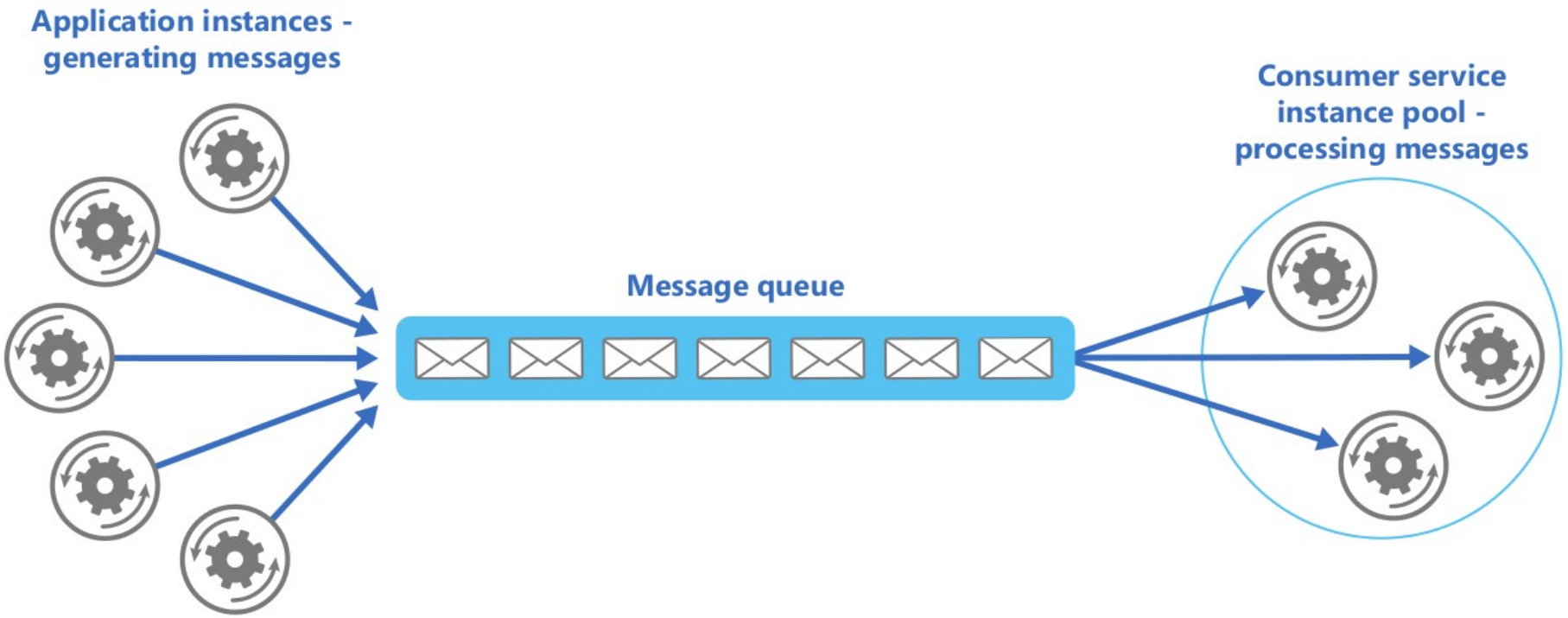
1: Application invokes operation on hosted service. The request fails, and the service host responds with HTTP response code 500 (internal server error).
2: Application waits for a short interval and tries again. The request still fails with HTTP response code 500.
3: Application waits for a longer interval and tries again. The request succeeds with HTTP response code 200 (OK).

Issues To Consider

- Short delayed repetitive retries on a server may degrade it, if it is running at maximum capacity
- If a request still fails after a certain amount of retries it may be best to stop emitting the request, here the Circuit Breaker comes in handy

Competing Consumer

- Enables multiple consumers to process messages sent on a shared channel concurrently
- Avoids the use of a synchronous medium to process requests


Application instances - generating messages
Message queue
Consumer service instance pool - processing messages

Compute Resource Consolidation

- Following the micro-service architecture, an application will compromise of a set of small services
- Grouping said services in one computational unit can reduce network overhead and running costs of the application

Issues To Consider

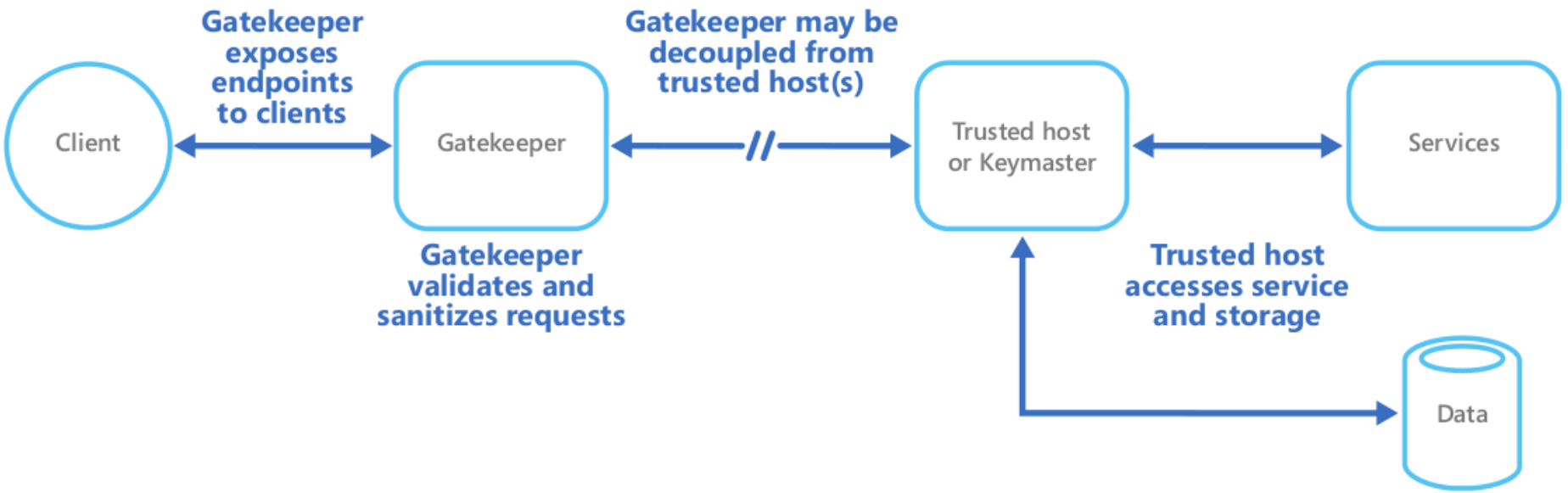
- **Not all services can be grouped together, it depends on their resource usage for instance:**
  - Grouping a database and a cache, would create a bottle neck in the system as all the requests would have to go to one unit
  - Grouping a cache and a messaging queue would cause race conditions on the memory

External Configuration Store

- Store the configuration of the application in to a centralized location for easier management

Gatekeeper (Ambassador)

- Provides a Proxy for the clients to connect to, in order to provide a security layer
- Proxy checks each incoming connection from the client and only allows trusted connections to pass through to the backend
- It is important that the Gatekeeper does not have access to the keys used by the trusted host so that, if it is compromised, the attacker does not have access to the said keys

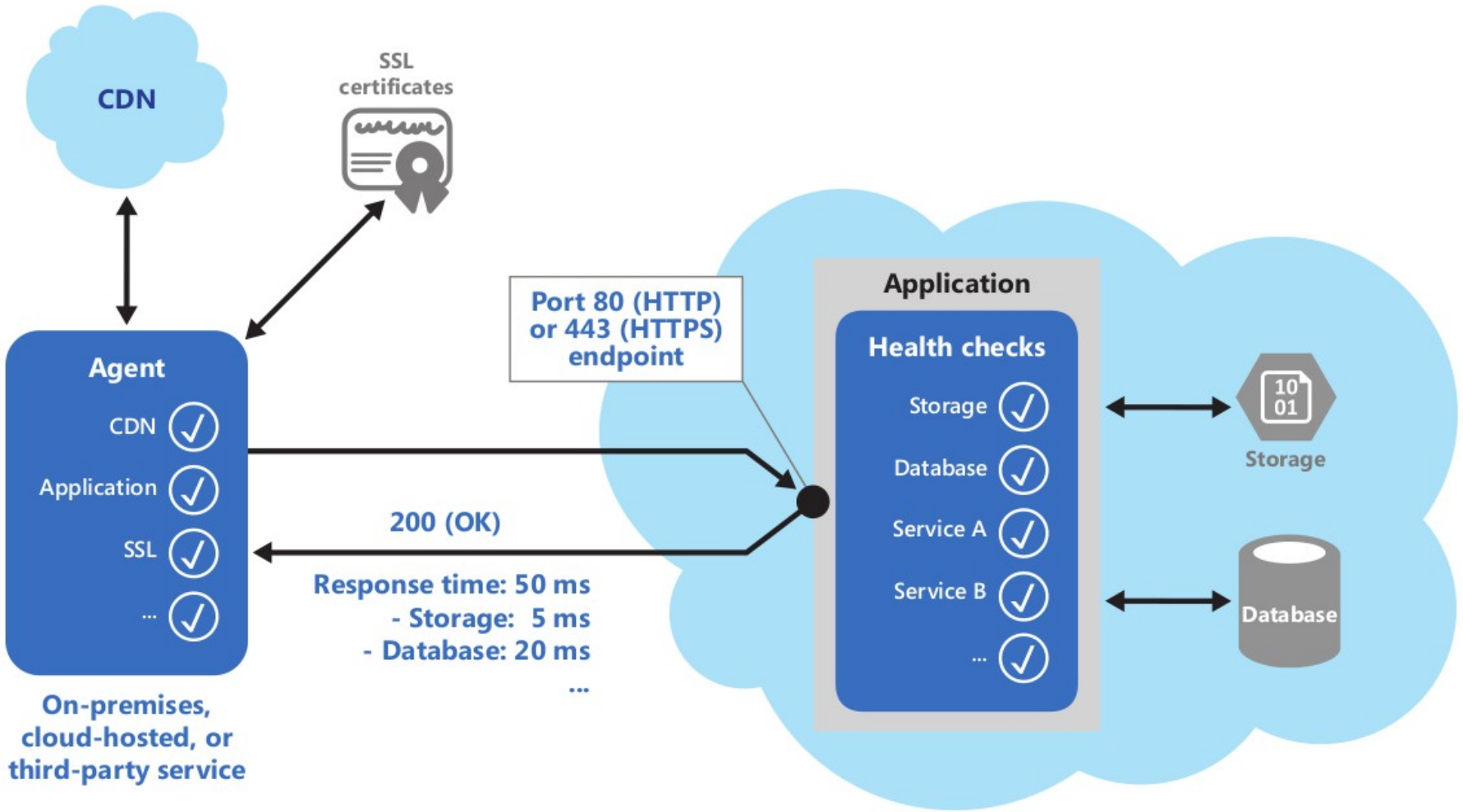

Gatekeeper exposes endpoints to clients
Gatekeeper may be decoupled from trusted host(s)
Client
Gatekeeper
Trusted host or Keymaster
Services
Gatekeeper validates and sanitizes requests
Trusted host accesses service and storage
Data

Issues To Consider

- The Gatekeeper should have access only to protected APIs from the host
- Gatekeeper must run in a limited privilege mode
- Gatekeeper and the trusted host should run in separate environments, e.g. separate Vms for each
- Gatekeepers could easily become a bottleneck in the system if not replicated enough
- Connection between Gatekeeper and trusted host should be encrypted

Health Endpoint Monitoring

- Provides an endpoint within an application so that external tools can verify the performance of the running application

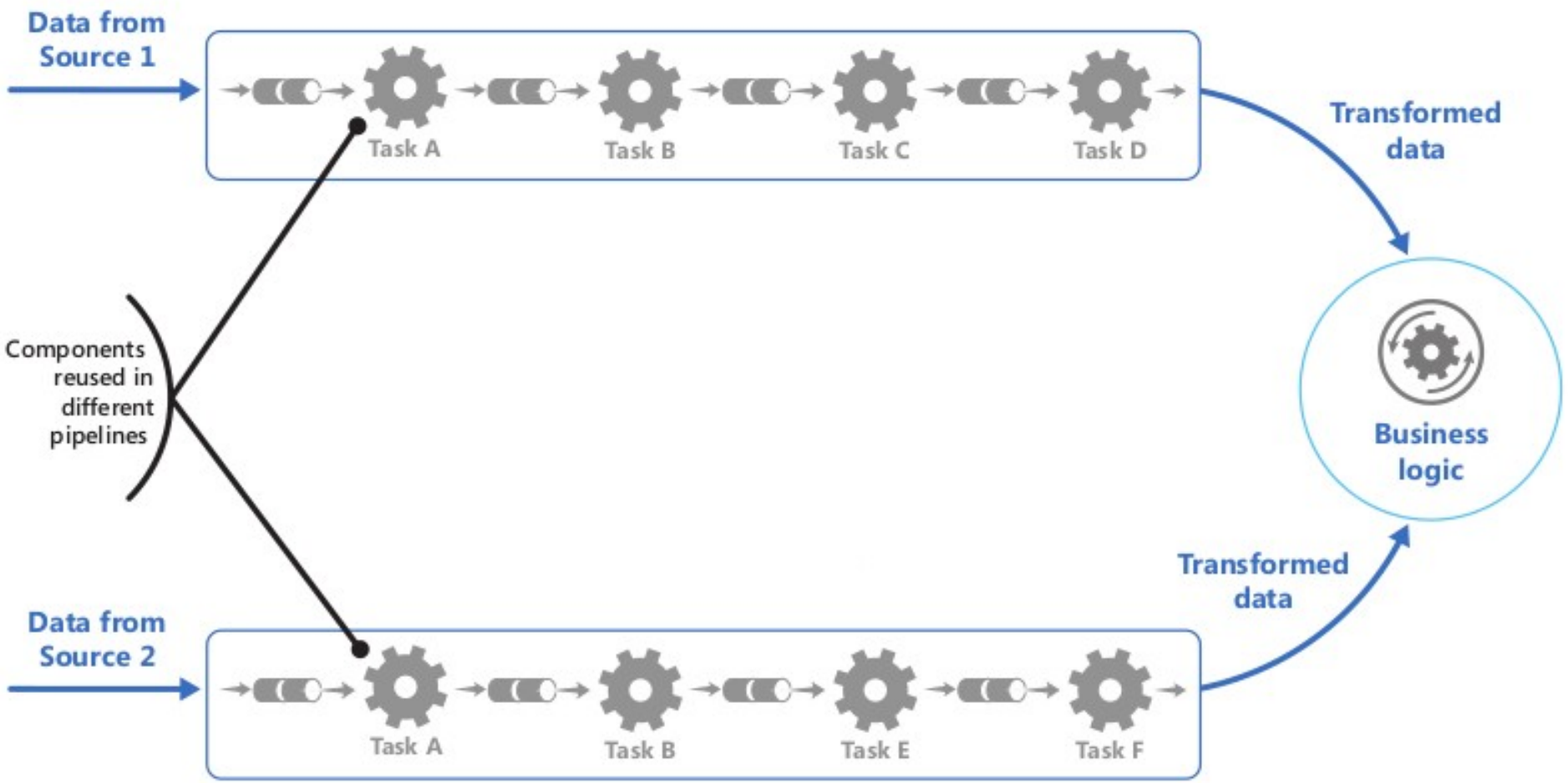

CDN
SSL certificates
Agent
CDN
Application
SSL
...
On-premises, cloud-hosted, or third-party service
Port 80 (HTTP) or 443 (HTTPS) endpoint
200 (OK)
Response time: 50 ms
- Storage: 5 ms
- Database: 20 ms
...
Application
Health checks
Storage
Database
Service A
Service B
...
10 01
Storage
Database

Issues To Consider

- What would be an accepted response that will validate the health of the application
- The number of endpoints to expose for health checks
- How extensive the test should be to avoid the application failing

Pipes And Filters

- Decompose applications into smaller applications (filters) that can be reused to form a pipeline of operations
- Allows for flexibility in adding operations to business logic

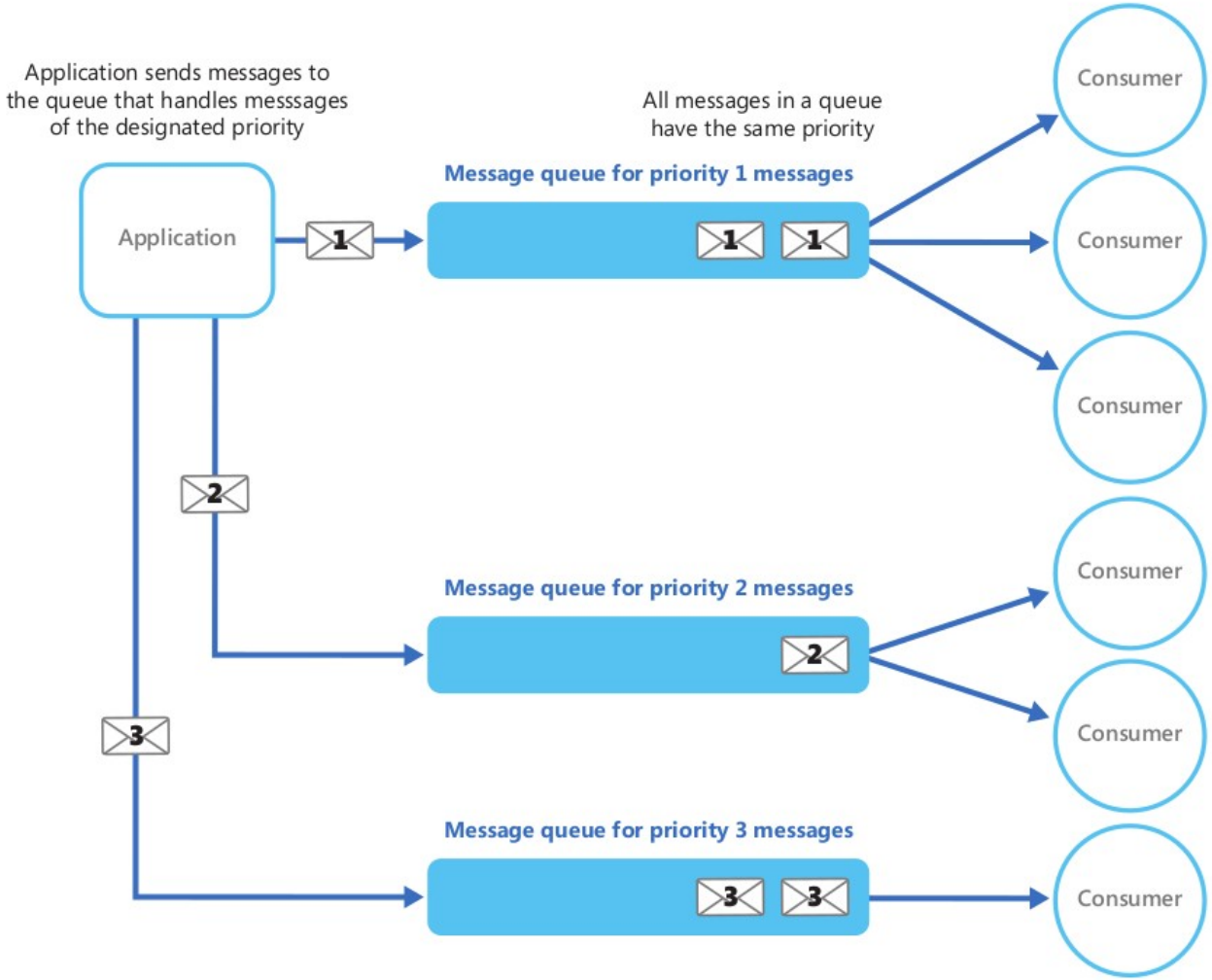

Data from Source 1
Task A
Task B
Task C
Task D
Transformed data
Components reused in different pipelines
Business logic
Transformed data
Data from Source 2
Task A
Task B
Task E
Task F

Issues To Consider

- The speed of the pipeline depends on the slowest filter
- Failure tolerance, if a filter fails, the tasks running on it should be rescheduled on another instance

Priority Queues

- Priortize requests send to a service so that they are processed faster
- Usually used in applications that offer different service levels to clients, such as premium and free users

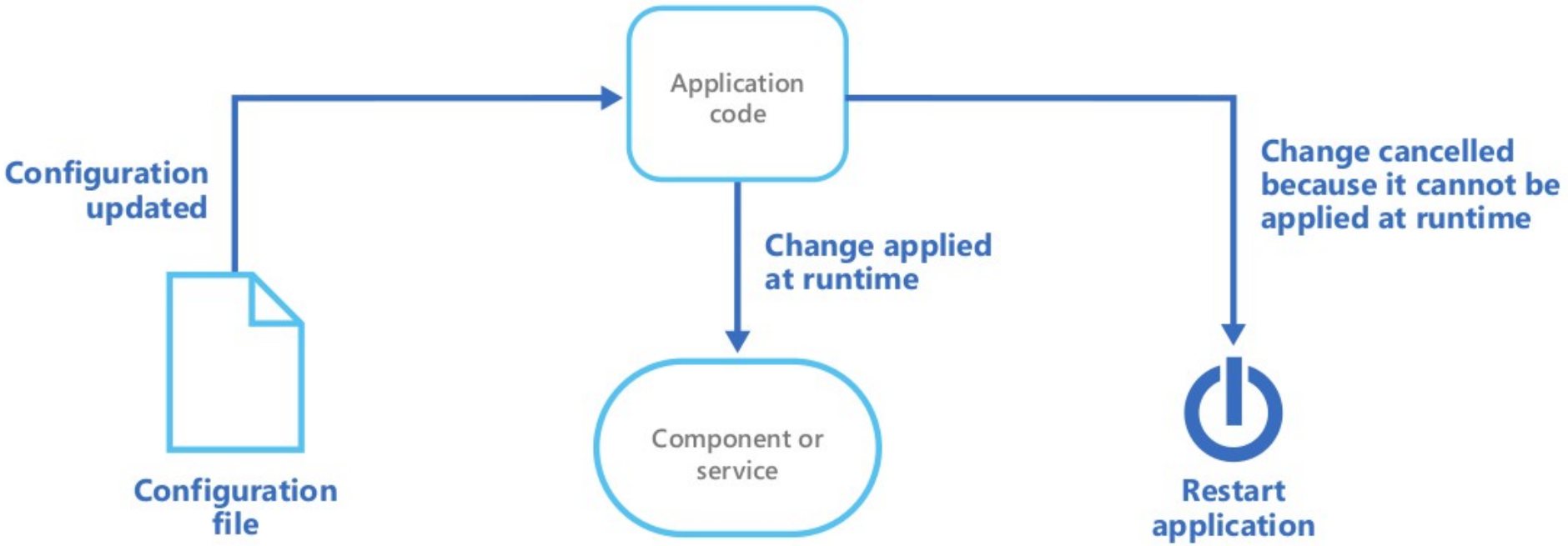

Application sends messages to the queue that handles messsages of the designated priority
All messages in a queue have the same priority
Application
Message queue for priority 1 messages
Message queue for priority 2 messages
Message queue for priority 3 messages
Consumer
Consumer
Consumer
Consumer
Consumer
Consumer

Issues To Consider

- You may want to allocate more resources to higher priority requests
- Using a separate queue for each message priority works best for a system with small number of pre-defined priorities

Runtime Reconfiguration

- Applications are designed so that they can be reconfigured without reloading the application
- Configuration files are written that define certain aspects of an application and are re-read on certain events, for example adding a new class, would append the class and the command it maps to in the file
- Used where downtime is expensive


Application code
Configuration updated
Configuration file
Change applied at runtime
Component or service
Change cancelled because it cannot be applied at runtime
Restart application

Issues To Consider

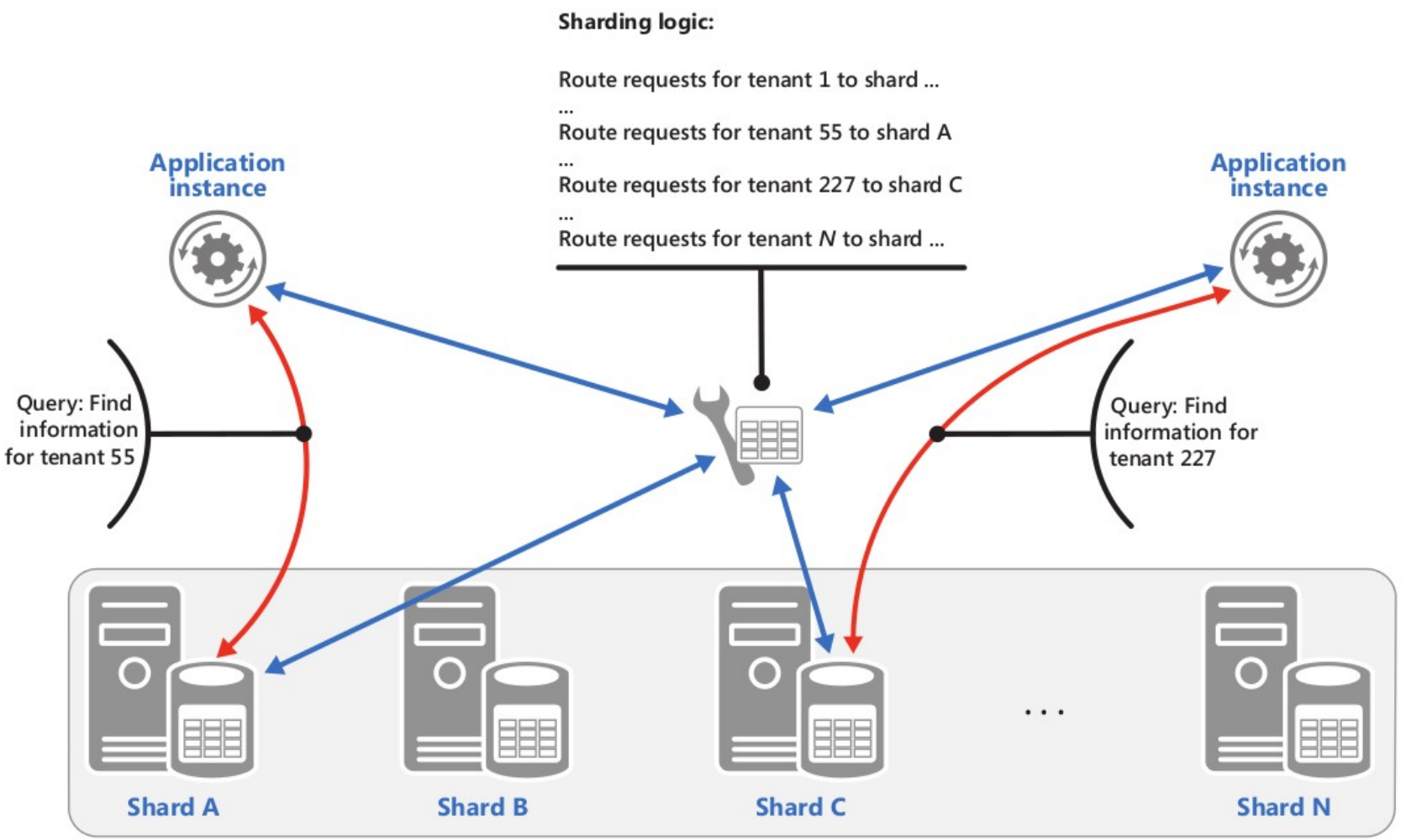
- Changes should be applied dynamically while application is running
- If the change cannot be applied dynamically, the application should handle restarting itself, either internally or by calling OS functions

Sharding

- **Data is divided across a set of horizontal partitions to serve more clients through the use of distributed resources, such as:**
  - Storage Space
  - Computational Power
  - Network Bandwidth
- **Allows for a more dynamic management of the data store, as now resources can be easily added to the cluster**
- **Three main strategies are used to shard**

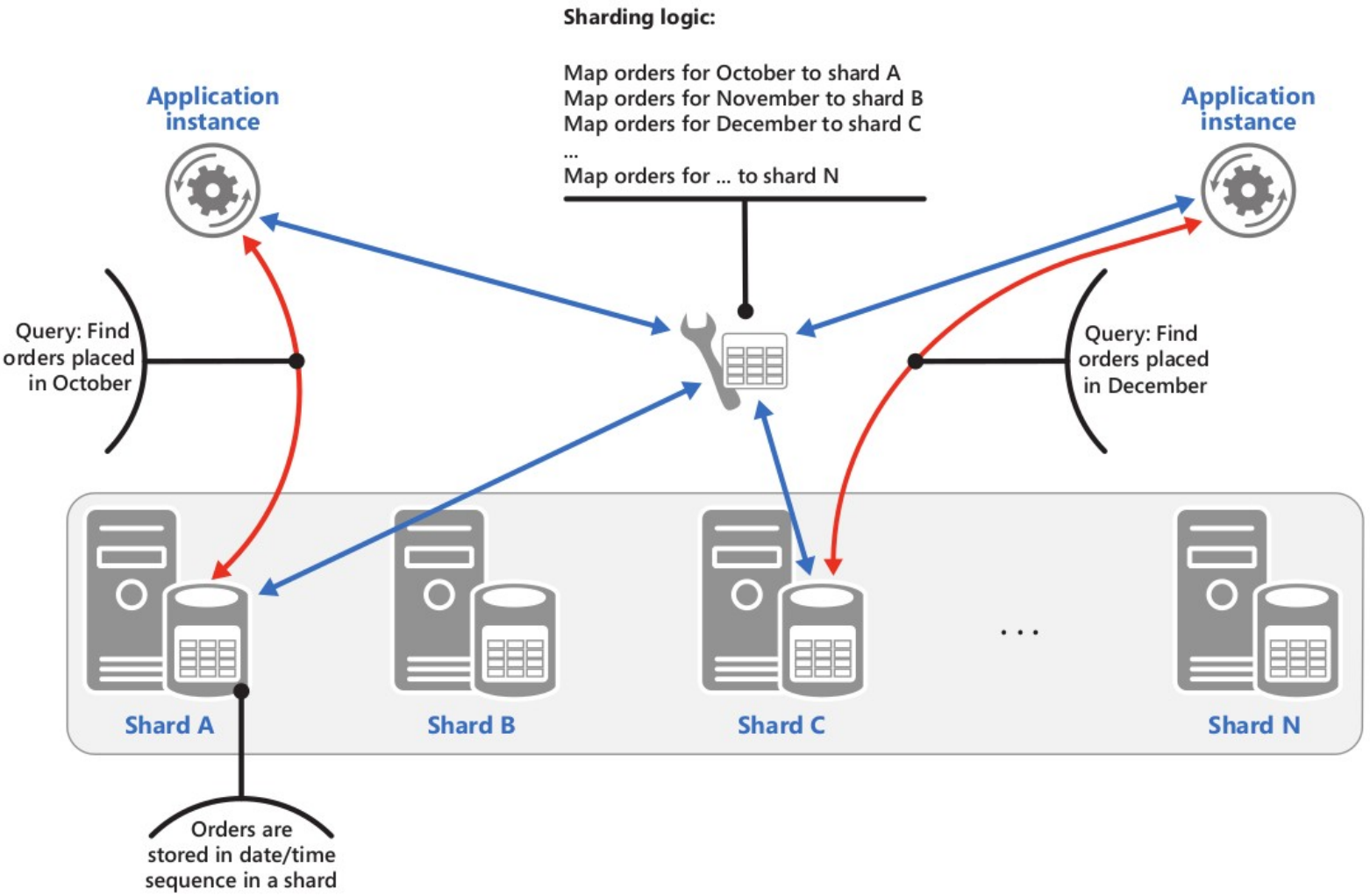
Sharding – Lookup Strategy

- A map is used to redirect requests to the correct shard containing the required data
- In an application with multiple clients, the data per client may be stored on one shard
- Shards can be shared with other clients


Sharding logic:
Route requests for tenant 1 to shard ...
...
Route requests for tenant 55 to shard A
...
Route requests for tenant 227 to shard C
...
Route requests for tenant N to shard ...
Application instance
Application instance
Query: Find information for tenant 55
Query: Find information for tenant 227
Shard A
Shard B
Shard C
Shard N

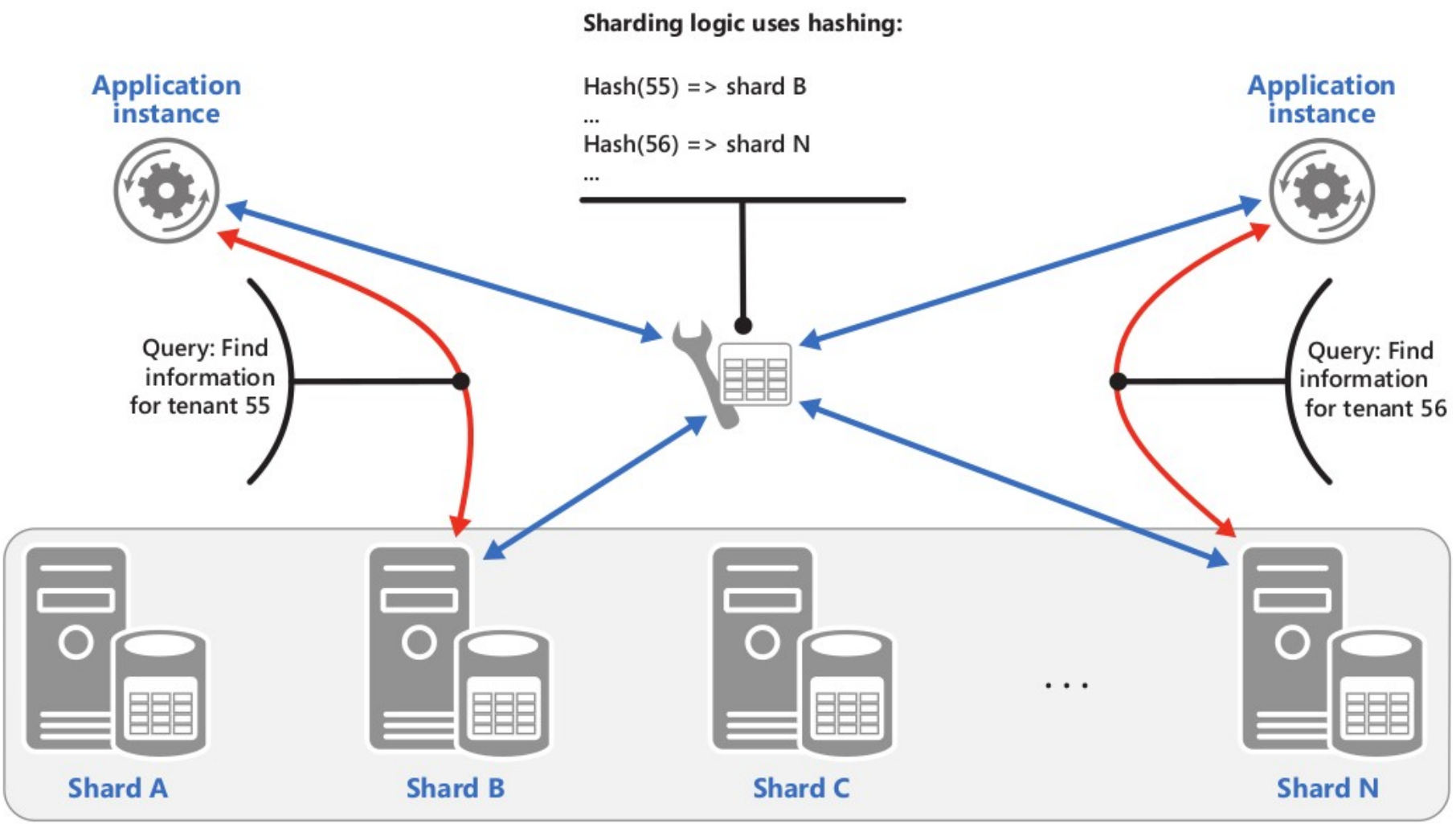
Sharding – Range Strategy

- Groups related items in the same shard and orders them by the shard key (primary key in the shard)
- More suited for applications that retrieve data using ranges, for example date ranges
- A map is used to redirect requests


Sharding logic:
Map orders for October to shard A
Map orders for November to shard B
Map orders for December to shard C
...
Map orders for ... to shard N
Application instance
Application instance
Query: Find orders placed in October
Query: Find orders placed in December
Shard A
Shard B
Shard C
Shard N
Orders are stored in date/time sequence in a shard

Sharding – Hashing Strategy

- **Tries to reduce hotspots by distributing the data across shards such that there is a balance between shard size and average load on shard**


Sharding logic uses hashing:
Hash(55) => shard B
...
Hash(56) => shard N
...
Application instance
Application instance
Query: Find information for tenant 55
Query: Find information for tenant 56
Shard A
Shard B
Shard C
Shard N

Issues To Consider

- Avoid schemas in which queries have to run on multiple shards to retrieve data, it is not possible to entirely removed them however
- Queries that have to run on multiple shards can be run in parallel
- Avoid using auto incrementing keys as the shard keys as they may not coordinate well across shards and may result in key conflicts

Static Content Hosting

- Static content are deployed on cloud based storage services rather than have them bundled with web servers
- This frees up the web server to handle other requests

Issues To Consider

- The storage service must offer an HTTP endpoint to service the files
- The storage services must be secured to avoid anyone writing new content in them