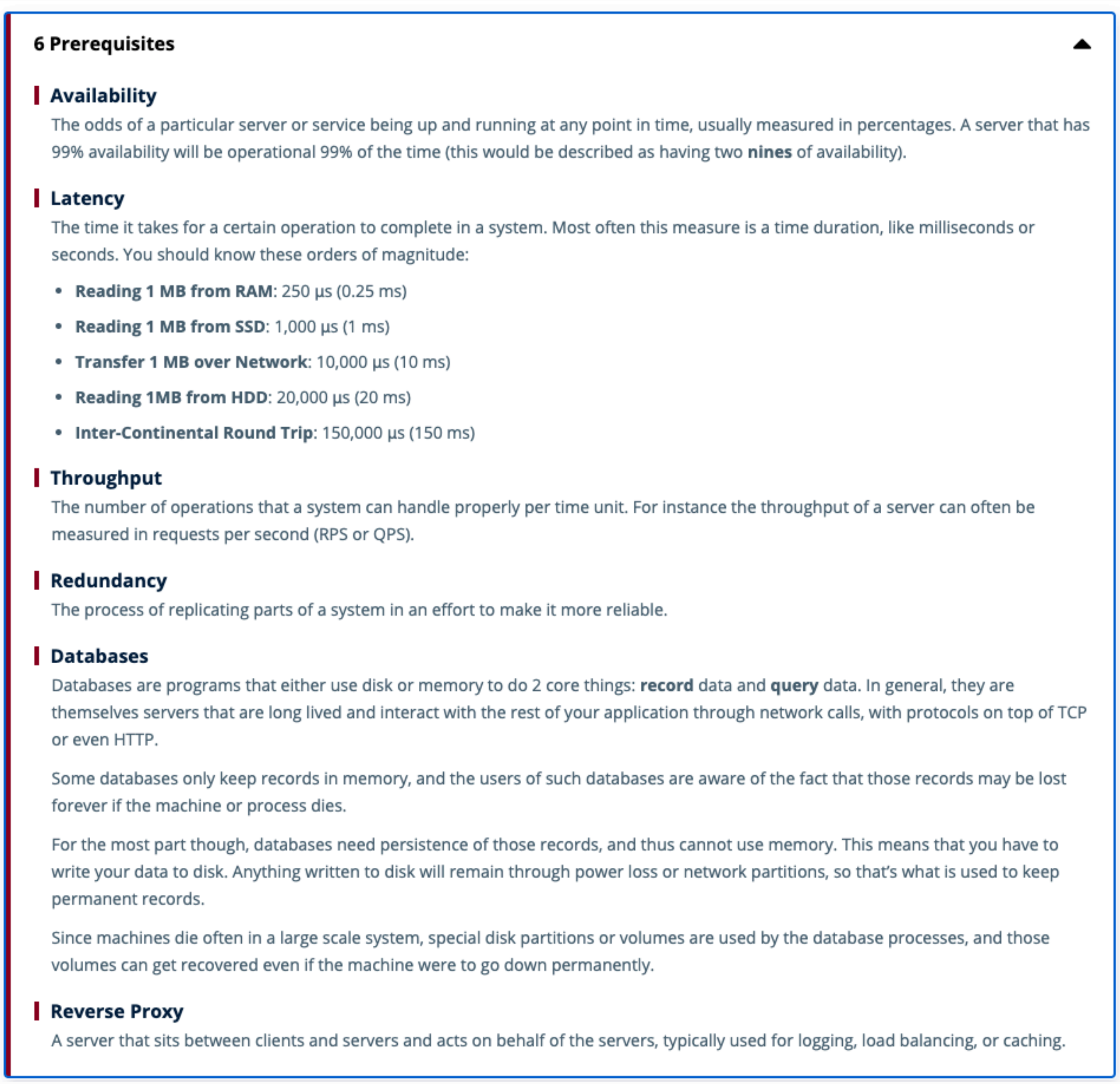
15_Replication_And_Sharding



6 Prerequisites

Availability

The odds of a particular server or service being up and running at any point in time, usually measured in percentages. A server that has 99% availability will be operational 99% of the time (this would be described as having two **nines** of availability).

Latency

The time it takes for a certain operation to complete in a system. Most often this measure is a time duration, like milliseconds or seconds. You should know these orders of magnitude:

- **Reading 1 MB from RAM**: 250 μs (0.25 ms)
- **Reading 1 MB from SSD**: 1,000 μs (1 ms)
- **Transfer 1 MB over Network**: 10,000 μs (10 ms)
- **Reading 1MB from HDD**: 20,000 μs (20 ms)
- **Inter-Continental Round Trip**: 150,000 μs (150 ms)

Throughput

The number of operations that a system can handle properly per time unit. For instance the throughput of a server can often be measured in requests per second (RPS or QPS).

Redundancy

The process of replicating parts of a system in an effort to make it more reliable.

Databases

Databases are programs that either use disk or memory to do 2 core things: **record** data and **query** data. In general, they are themselves servers that are long lived and interact with the rest of your application through network calls, with protocols on top of TCP or even HTTP.

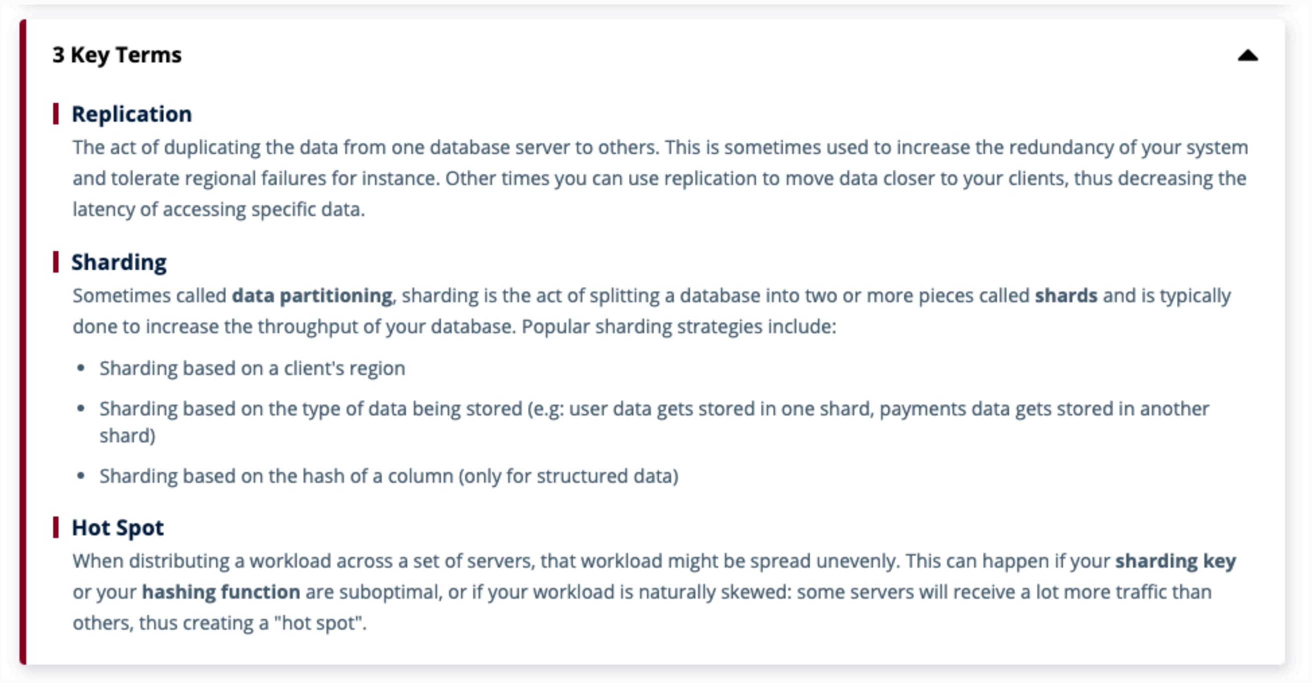
Some databases only keep records in memory, and the users of such databases are aware of the fact that those records may be lost forever if the machine or process dies.

For the most part though, databases need persistence of those records, and thus cannot use memory. This means that you have to write your data to disk. Anything written to disk will remain through power loss or network partitions, so that's what is used to keep permanent records.

Since machines die often in a large scale system, special disk partitions or volumes are used by the database processes, and those volumes can get recovered even if the machine were to go down permanently.

Reverse Proxy

A server that sits between clients and servers and acts on behalf of the servers, typically used for logging, load balancing, or caching.



3 Key Terms

Replication

The act of duplicating the data from one database server to others. This is sometimes used to increase the redundancy of your system and tolerate regional failures for instance. Other times you can use replication to move data closer to your clients, thus decreasing the latency of accessing specific data.

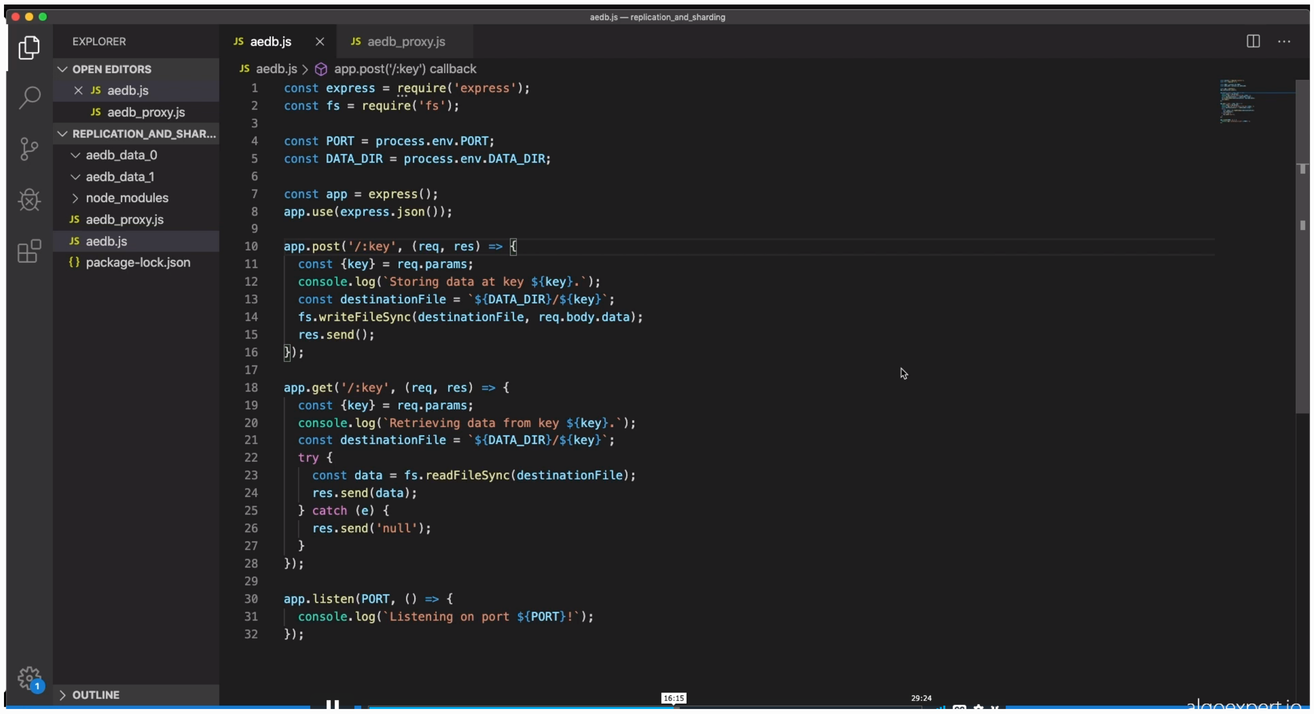
Sharding

Sometimes called **data partitioning**, sharding is the act of splitting a database into two or more pieces called **shards** and is typically done to increase the throughput of your database. Popular sharding strategies include:

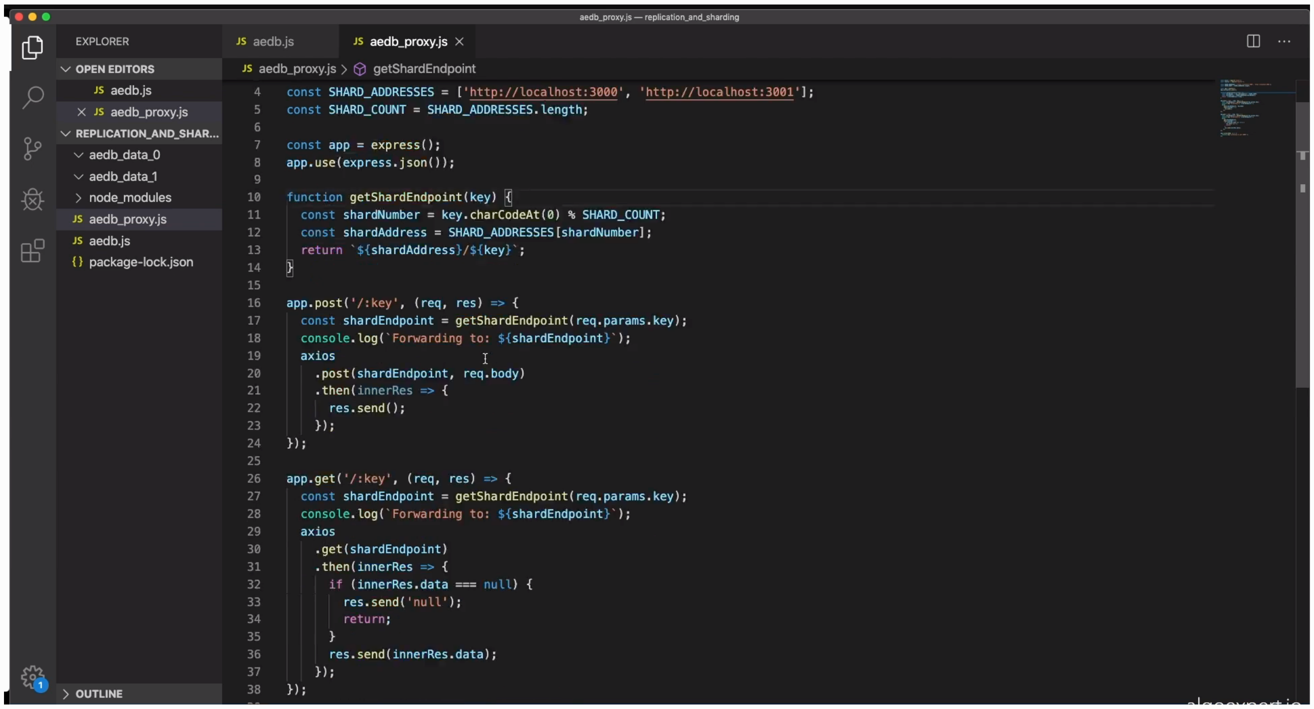
- Sharding based on a client's region
- Sharding based on the type of data being stored (e.g: user data gets stored in one shard, payments data gets stored in another shard)
- Sharding based on the hash of a column (only for structured data)

Hot Spot

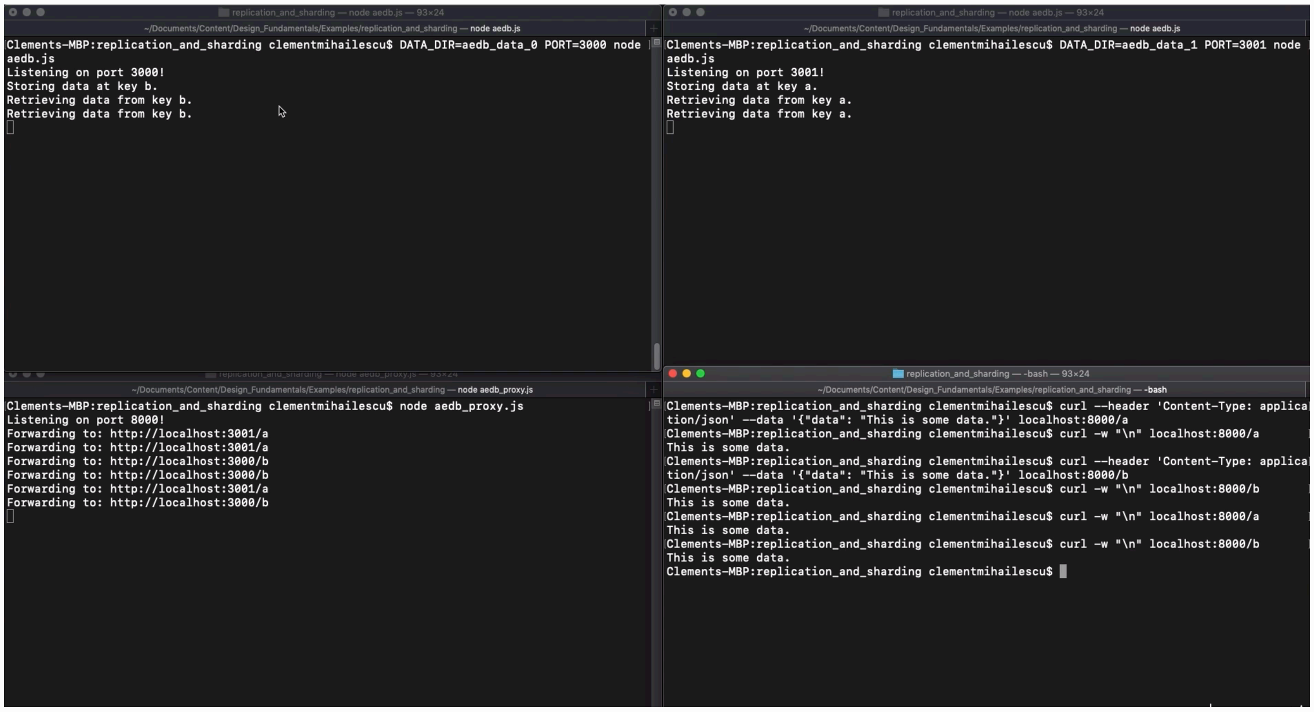
When distributing a workload across a set of servers, that workload might be spread unevenly. This can happen if your **sharding key** or your **hashing function** are suboptimal, or if your workload is naturally skewed: some servers will receive a lot more traffic than others, thus creating a "hot spot".



```javascript
const express = require('express');
const fs = require('fs');

const PORT = process.env.PORT;
const DATA_DIR = process.env.DATA_DIR;

const app = express();
app.use(express.json());

app.post('/:key', (req, res) => {
  const {key} = req.params;
  console.log(`Storing data at key ${key}.`);
  const destinationFile = `${DATA_DIR}/${key}`;
  fs.writeFileSync(destinationFile, req.body.data);
  res.send();
});

app.get('/:key', (req, res) => {
  const {key} = req.params;
  console.log(`Retrieving data from key ${key}.`);
  const destinationFile = `${DATA_DIR}/${key}`;
  try {
    const data = fs.readFileSync(destinationFile);
    res.send(data);
  } catch (e) {
    res.send('null');
  }
});

app.listen(PORT, () => {
  console.log(`Listening on port ${PORT}!`);
});
```



```javascript
const SHARD_ADDRESSES = ['http://localhost:3000', 'http://localhost:3001'];
const SHARD_COUNT = SHARD_ADDRESSES.length;

const app = express();
app.use(express.json());

function getShardEndpoint(key) {
  const shardNumber = key.charCodeAt(0) % SHARD_COUNT;
  const shardAddress = SHARD_ADDRESSES[shardNumber];
  return `${shardAddress}/${key}`;
}

app.post('/:key', (req, res) => {
  const shardEndpoint = getShardEndpoint(req.params.key);
  console.log(`Forwarding to: ${shardEndpoint}`);
  axios
    .post(shardEndpoint, req.body)
    .then(innerRes => {
      res.send();
    });
});

app.get('/:key', (req, res) => {
  const shardEndpoint = getShardEndpoint(req.params.key);
  console.log(`Forwarding to: ${shardEndpoint}`);
  axios
    .get(shardEndpoint)
    .then(innerRes => {
      if (innerRes.data === null) {
        res.send('null');
        return;
      }
      res.send(innerRes.data);
    });
});
```


replication_and_sharding — node aedb.js — 93×24
~/Documents/Content/Design_Fundamentals/Examples/replication_and_sharding — node aedb.js
replication_and_sharding — node aedb.js — 93×24
~/Documents/Content/Design_Fundamentals/Examples/replication_and_sharding — node aedb.js
replication_and_sharding — node aedb_proxy.js — 93×24
~/Documents/Content/Design_Fundamentals/Examples/replication_and_sharding — node aedb_proxy.js
replication_and_sharding — -bash — 93×24
~/Documents/Content/Design_Fundamentals/Examples/replication_and_sharding — -bash

```
Clements-MBP:replication_and_sharding clementmihailescu$ DATA_DIR=aedb_data_0 PORT=3000 node
aedb.js
Listening on port 3000!
Storing data at key b.
Retrieving data from key b.
Retrieving data from key b.
```

```
Clements-MBP:replication_and_sharding clementmihailescu$ DATA_DIR=aedb_data_1 PORT=3001 node
aedb.js
Listening on port 3001!
Storing data at key a.
Retrieving data from key a.
Retrieving data from key a.
```

```
Clements-MBP:replication_and_sharding clementmihailescu$ node aedb_proxy.js
Listening on port 8000!
Forwarding to: http://localhost:3001/a
Forwarding to: http://localhost:3001/a
Forwarding to: http://localhost:3000/b
Forwarding to: http://localhost:3000/b
Forwarding to: http://localhost:3001/a
Forwarding to: http://localhost:3000/b
```

```
Clements-MBP:replication_and_sharding clementmihailescu$ curl --header 'Content-Type: applica
tion/json' --data '{"data": "This is some data."}' localhost:8000/a
Clements-MBP:replication_and_sharding clementmihailescu$ curl -w "\n" localhost:8000/a
This is some data.
Clements-MBP:replication_and_sharding clementmihailescu$ curl --header 'Content-Type: applica
tion/json' --data '{"data": "This is some data."}' localhost:8000/b
Clements-MBP:replication_and_sharding clementmihailescu$ curl -w "\n" localhost:8000/b
This is some data.
Clements-MBP:replication_and_sharding clementmihailescu$ curl -w "\n" localhost:8000/a
This is some data.
Clements-MBP:replication_and_sharding clementmihailescu$ curl -w "\n" localhost:8000/b
This is some data.
Clements-MBP:replication_and_sharding clementmihailescu$
```