Backend API Design for SREs

Sam Dunster 🇦🇺
Production Engineer
sdunster@meta.com

Meta

APIs are everywhere

Public APIs

Public APIs

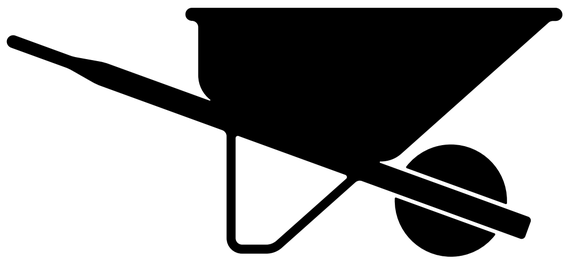
Backend APIs

Backend API design:

Why is it important?

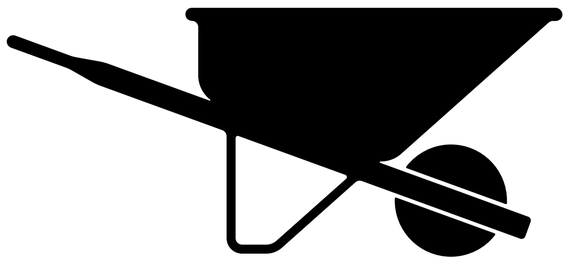
Backend API design:

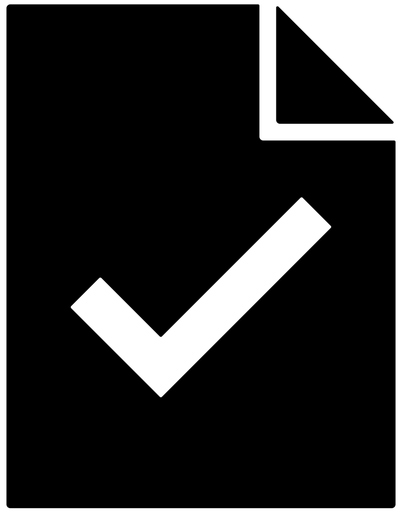
Why is it important?



Backend API design:

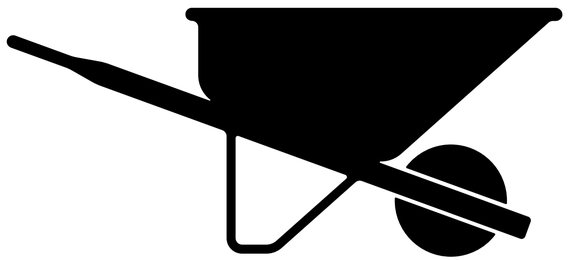
Why is it important?

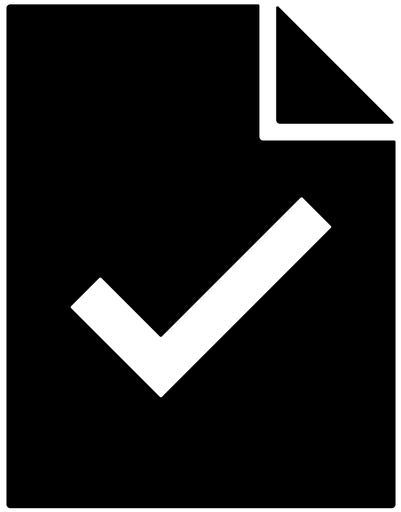




Backend API design:

Why is it important?





5 tips for designing good backend APIs

Ensure you speak the same language

vol:123

Sam’s Wonderful Volume

142BA58C-311E-443A-9159-683B1493AFFB

my_volume

Naming is hard

7492574

ZippyDB Shard 576 Volume

h6fMB4Kp

service_a_volume

```
CREATE TABLE volumes (
   `id` int(20) NOT NULL AUTO_INCREMENT,
   PRIMARY KEY (`id`)
);
```

```
CREATE TABLE volumes (
  `id` varchar(255) NOT NULL,
  PRIMARY KEY (`id`)
);
```

142BA58C-311E-443A-9159-683B1493AFFB


```
$ vol mount mel1:142BA58C-311E-443A-9159-683B1493AFFB
$ vol mount bne2:1868767474
$ vol mount syd1:my_volume
$ vol mount mel2:6Ae2Fm
```

https://github.com/ai/nanoid

```
use nanoid::nanoid;

fn main() {
    nanoid!(10, &nanoid::alphabet::SAFE); //=> "93ce_Ltuub"
}
```

Nano ID Collision Calculator

Nano ID is a library for generating random IDs. Likewise UUID, there is a probability of duplicate IDs. However, this probability is extremely small.

Meanwhile, a lot of projects generate IDs in small numbers. For those projects, the ID length could be reduced without risk.

This calculator aims to help you realize the extent to which the ID length can be reduced.

Alphabet:

0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ_abcdefghijklmnopqrstuvwxyz-

64/256

ID length: 12 characters

Speed: 1000 IDs per hour/second

~1 thousand years needed, in order to have a 1% probability of at least one collision.

https://zelark.github.io/nano-id-cc/

```
$ vol mount mel2:6Ae2Fm
                      6Ae2Fm
                      6Ae2Fm
                      68Tdb5
                      6Ae2Fm
```

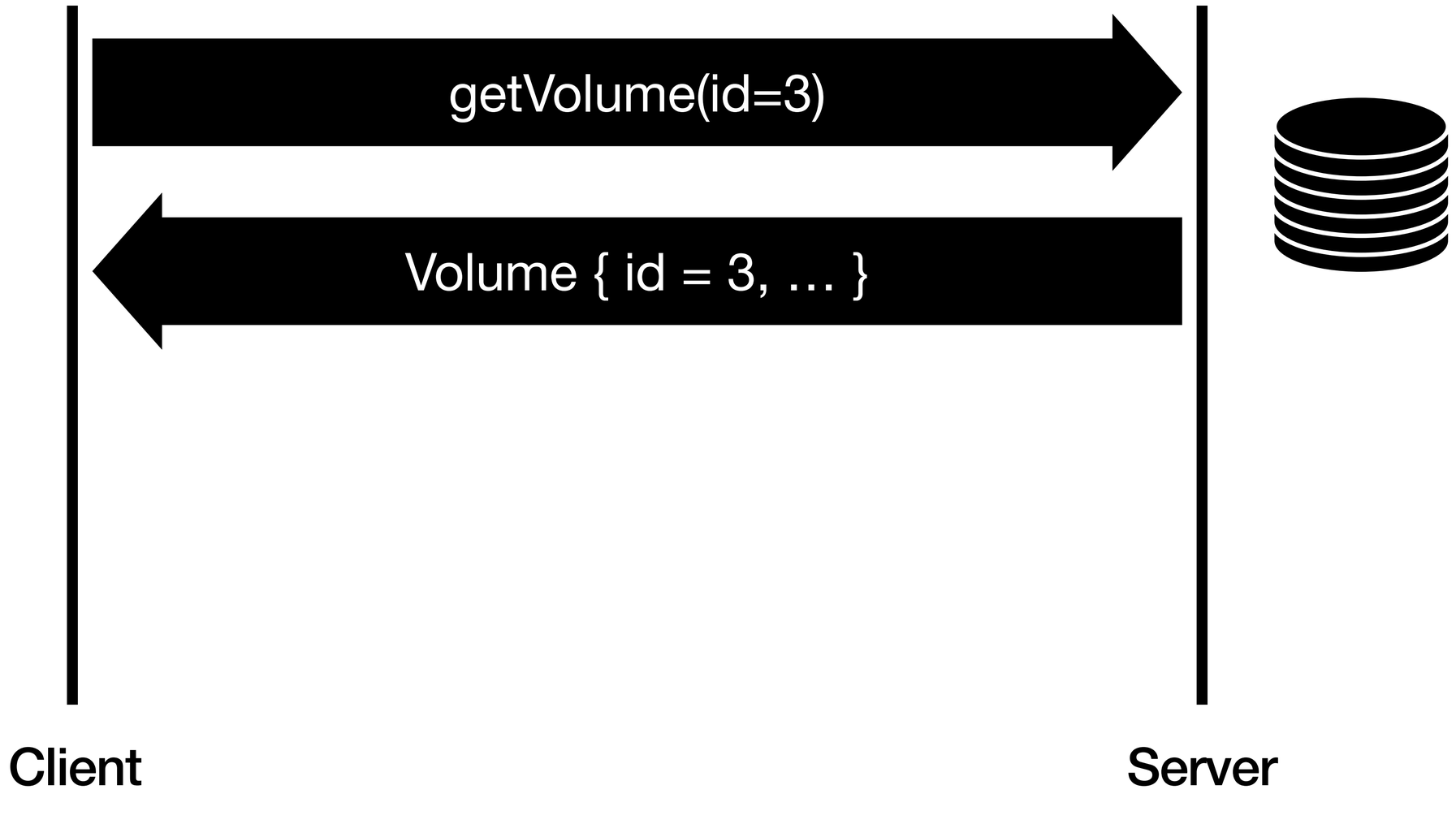
Volume ID:

6Ae2Fm

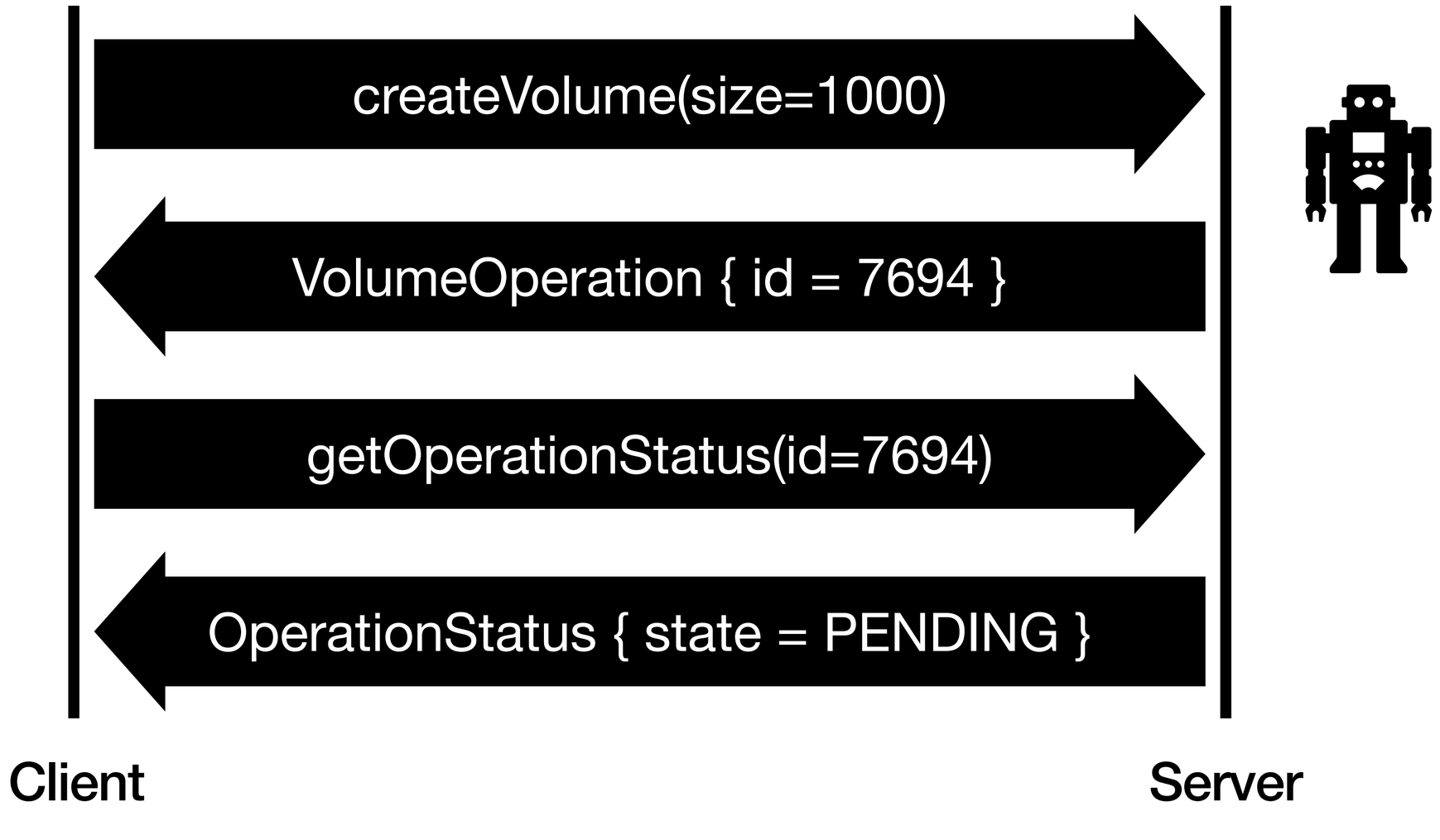
Plan ahead for potentially asynchronous operations

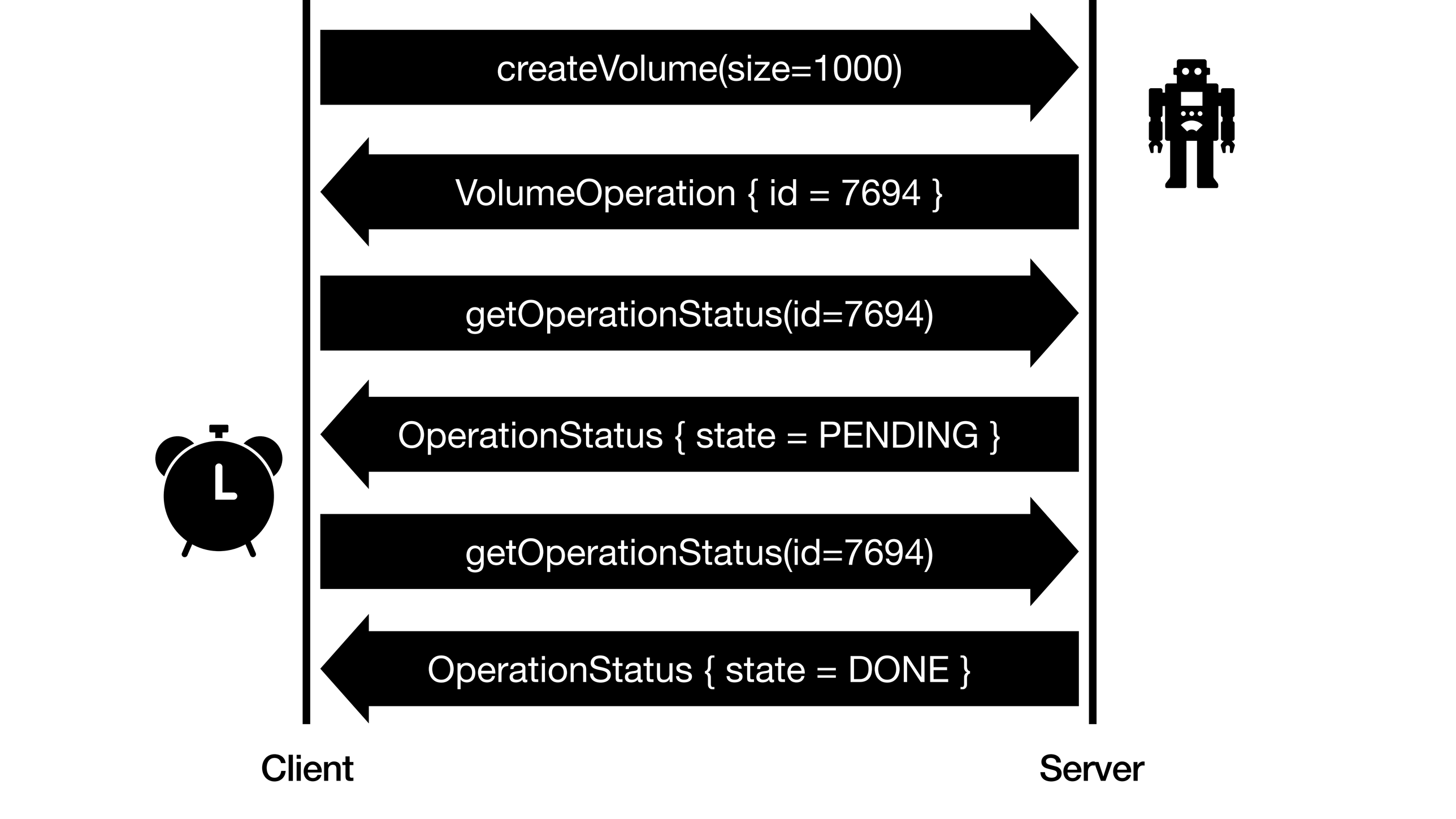
synchronous vs asynchronous

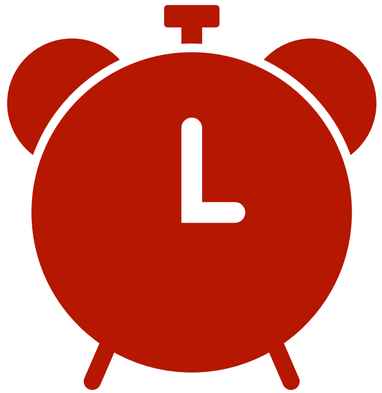
synchronous

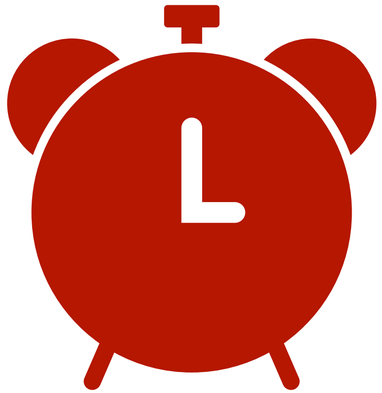


asynchronous

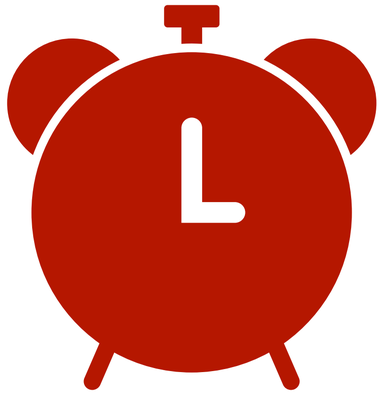



createVolume(size=1000)
VolumeOperation { id = 7694 }
getOperationStatus(id=7694)
OperationStatus { state = PENDING }
getOperationStatus(id=7694)
OperationStatus { state = DONE }
Client
Server





< 30 sec



< 30 sec

Create

Read

Update

Delete

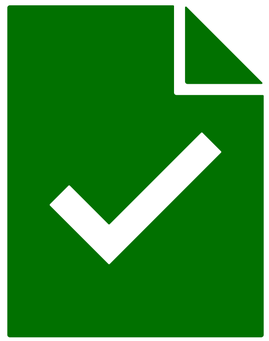
mountVolume()
takeSnapshot()
startVM()

Why not make **everything** asynchronous?

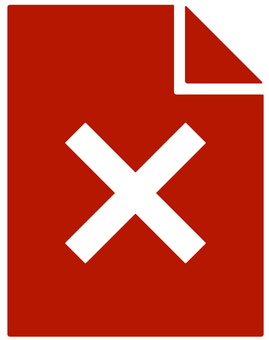
Differentiate between user and infrastructure errors

```
struct Volume {
  1: i32 id;
  2: i64 size;
}

Volume getVolume(
  1: i32 id,
) throws (
  1: GetVolumeExn ex,
);
```

```
struct Volume {
  1: i32 id;
  2: i64 size;
}

Volume getVolume(
  1: i32 id,
) throws (
  1: GetVolumeExn ex,
);
```

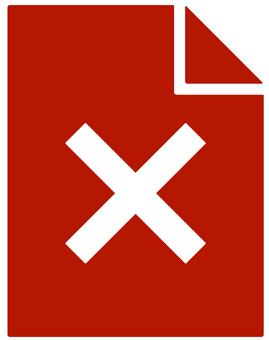


Volume



GetVolumeExn

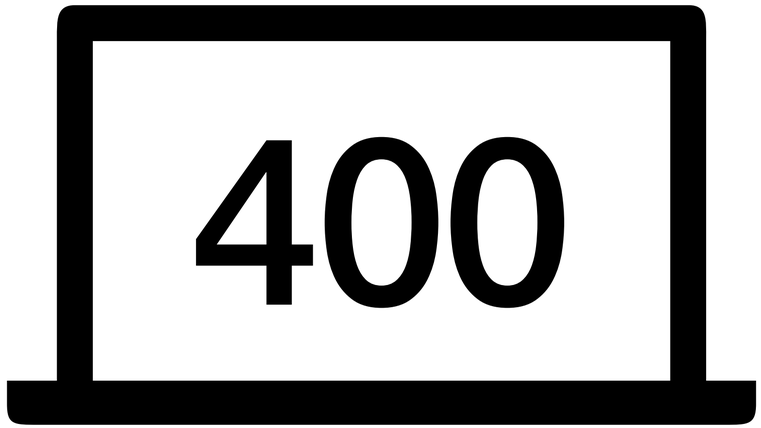
```
getVolume(100)
```



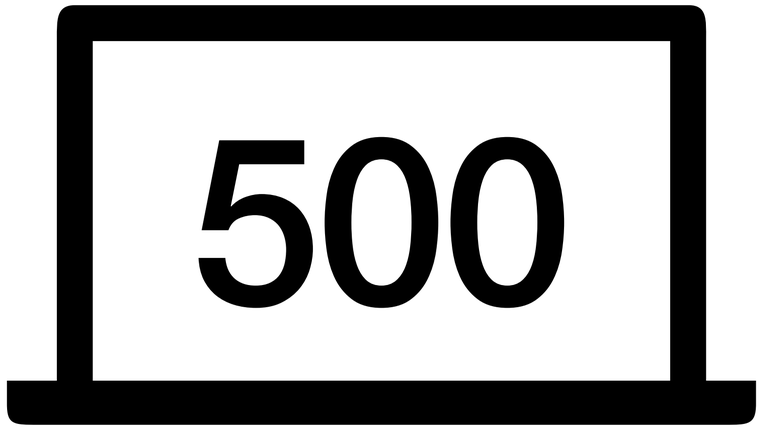
GetVolumeExn

HTTP

HTTP



Client error



Infra error

Client error

Object not found

Invalid parameters/validation error

Permission denied/no access

Infra error

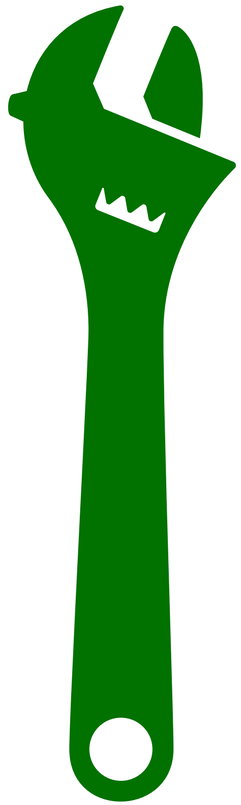
Failed to connect to the database

Service overloaded/thottling

Misconfiguration

Service unavailable due
to restart/upgrade

Client error



Infra error

Write high quality API documentation

```
struct Volume {
  1: i32 id;
  2: i64 size;
}

service DirectorService extends FacebookService {
  Volume getVolume(
    1: i32 id,
  ) throws (
    1: GetVolumeExn ex,
  );
}
```

```
service DirectorService extends FacebookService {
  void createVolume(
    1: optional i64 id,
    2: optional string name,
    3: i64 size,
  );
}
```

```
service DirectorService extends FacebookService {
  void createVolume(
    // the ID for the volume
    1: optional i64 id,
    // the name for the volume
    2: optional string name,
    // the size for the volume
    3: i64 size,
  );
}
```

```
service DirectorService extends FacebookService {
  void createVolume(
    // pass `id` or `name` but not both
    // omit both to generate an ID
    1: optional i64 id,
    2: optional string name,
    3: i64 size,  // MiB
  );
}
```

```
service DirectorService extends FacebookService {
  void updateVolume(
    1: optional i64 id,
    2: optional string name,
    3: optional i64 size,
  );
}
```

Test your API, use it yourself

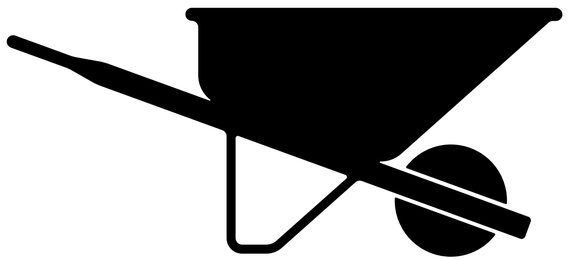
Integration tests

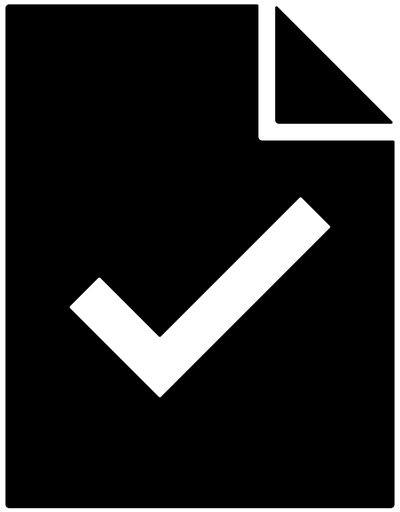
Eat your own dog food


> CLI

Backend API design:

Why is it important?





1. Ensure you speak the same language
2. Plan ahead for potentially asynchronous operations
3. Differentiate between user and infrastructure errors
4. Write high quality API documentation
5. Test your API, use it yourself

Sam Dunster

Email: sdunster@meta.com

Messenger: m.me/sdunster

SREcon Slack: #22apac-day3-track1