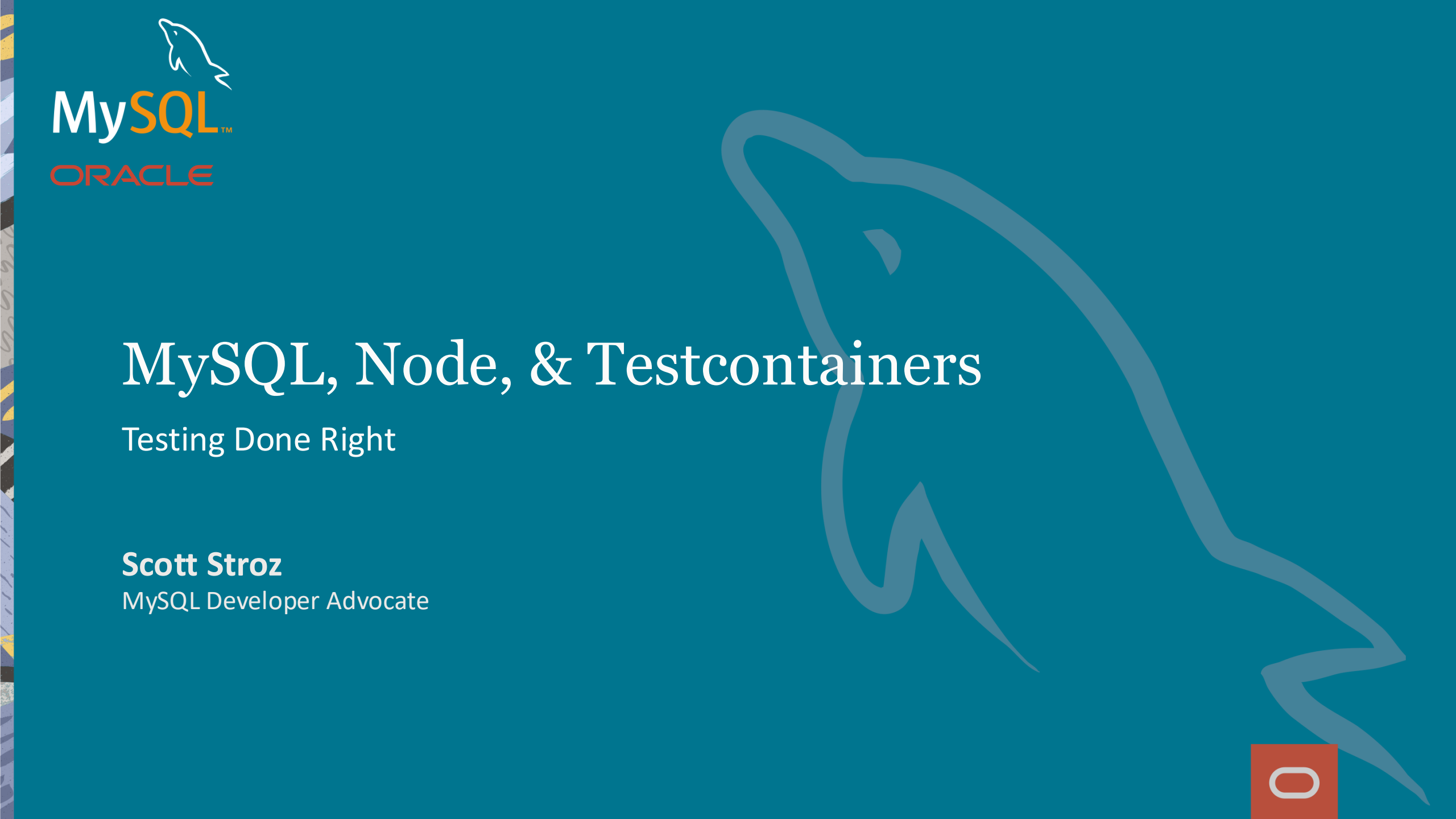


MySQL, Node, & Testcontainers

Testing Done Right

Scott Stroz
MySQL Developer Advocate

Obligatory "I Love Me" Slide

- Full-stack developer before we were called "full-stack" developers
- The only constant in my development stack has been MySQL
- Die-hard NY Giants Fan
- I've become smitten with writing tests for code
- I have the best office mate!

Why bother with testing code?

- Find bugs faster
- Helps in debugging
- Can assist in reducing code complexity
- Improved code quality
- Adjunct to documentation

Unit Tests vs. Integration Tests

Unit Tests

- Focused on a single function or method.
  - Testing in isolation

Image by LoggaWiggler from Pixabay

Unit Tests vs. Integration Tests

Integration Tests

- Tests complete, end-to-end processes
- Verifies that the different parts of the application work together as expected.

Photo by Jonny Gios on Unsplash

Why is Testing With a DB Challenging?

- Testing DB interaction is a combo of unit and integration testing (from a certain point of view)
- Database testing should be done on an island
  - The database may not be approved to run locally for testing

Photo by Sean Benesh on Unsplash

Testcontainers to the Rescue!

"Testcontainers is an open-source framework for providing throwaway, lightweight instances of databases, message brokers, web browsers, or just about anything that can run in a Docker container."

- Testcontainers Website

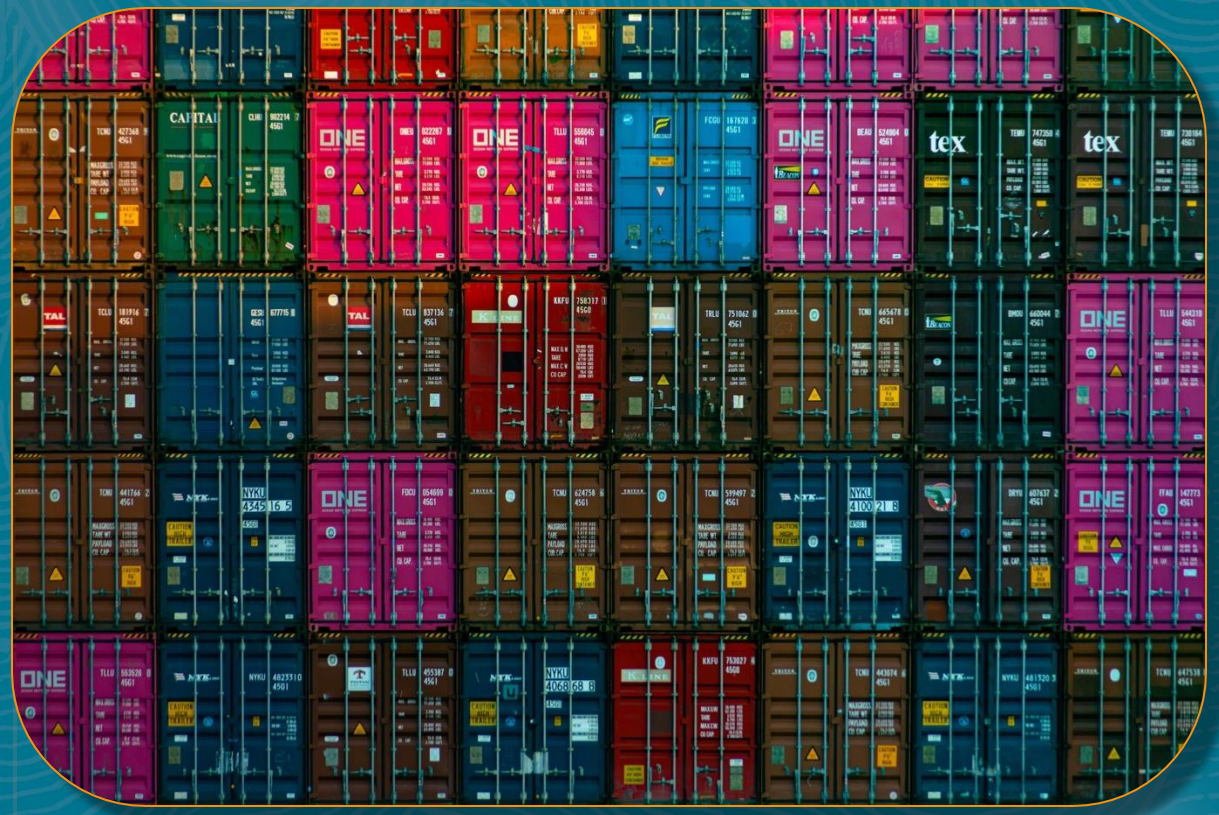
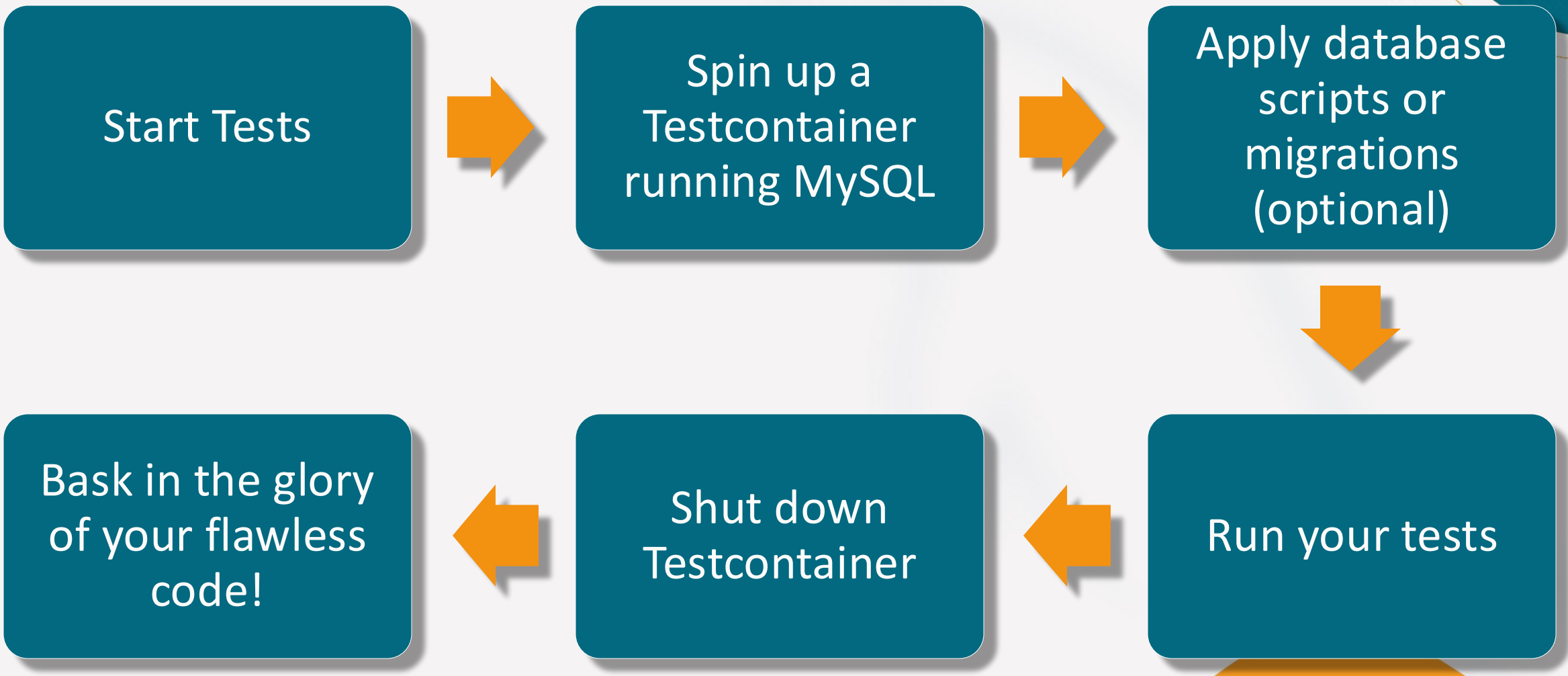
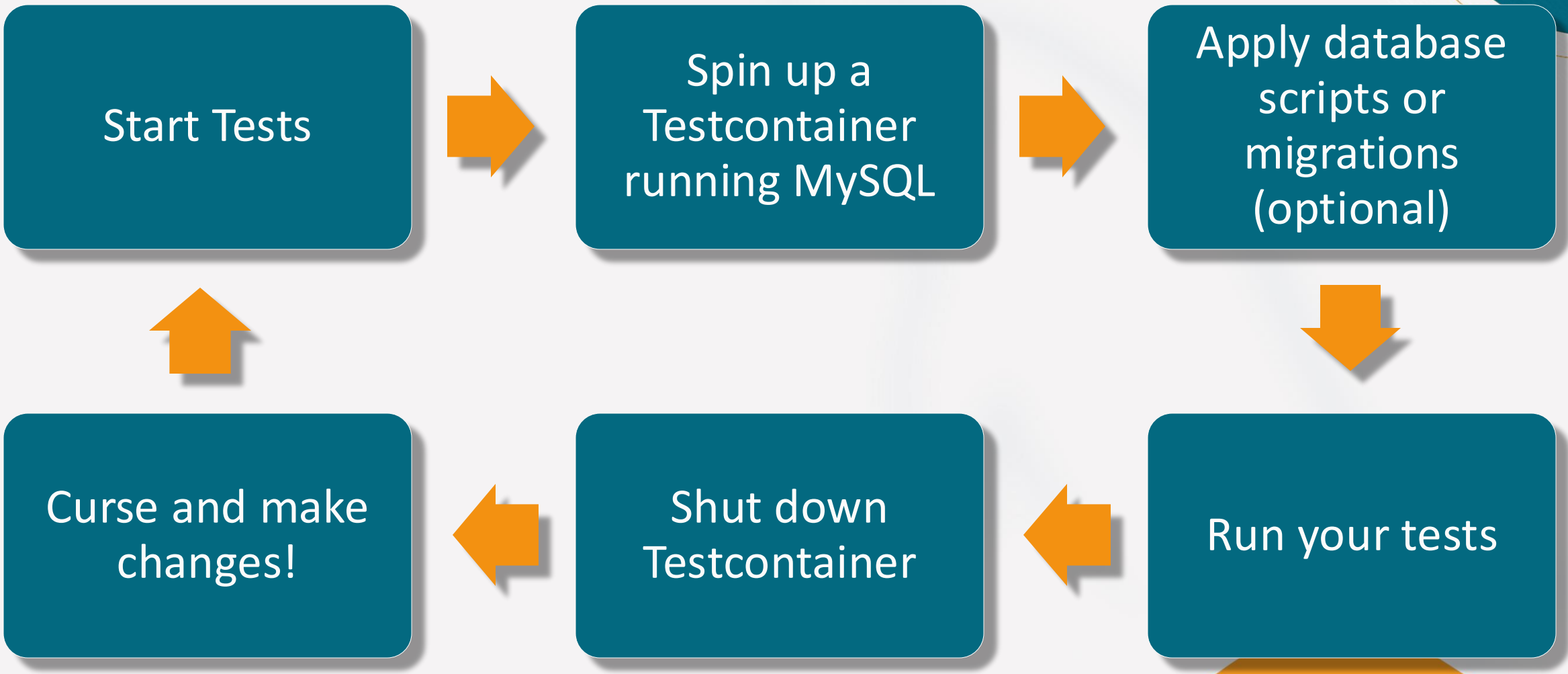


Photo by Teng Yuhong on Unsplash

How it Should Work



How it Should Work

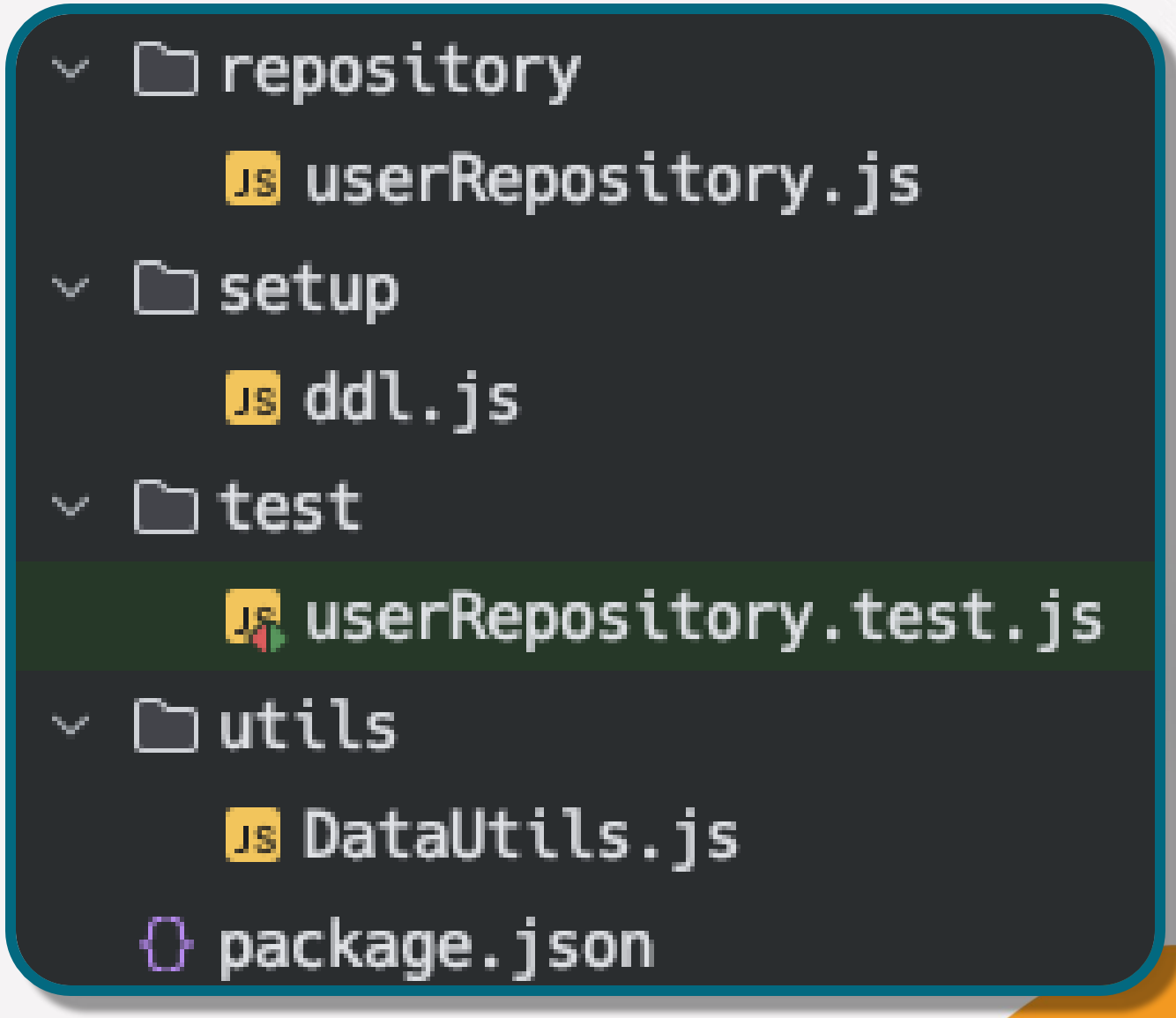


Node Test Runner

- Built-in Test Runner
  - Version 18 or higher
- Eliminates the need for third-party dependencies
- Supports similar syntax and structure of Mocha, Jasmine, and Jest.

Photo by Josh Gordon on Unsplash

Project Structure



Our Test File

```
import { test, describe, before, after } from 'node:test'
import { strict as assert } from 'node:assert'
import {MySqlContainer} from "@testcontainers/mysql"
import { faker } from '@faker-js/faker'
import { ddl } from "../setup/ddl.js"
import UserRepo from "../repository/userRepository.js"
import DataUtils from "../utils/DataUtils.js"

describe('Scott\'s Amazing Test Demo!!', async (t) => {
    let container
1   let userRepo
    let dataUtils

2   before(async ()=>{
3     container = await new MySqlContainer()
            .withExposedPorts(3306, 33060)
            .start();

4     userRepo = new UserRepo(
            container.getUsername(),
            container.getUserPassword(),
            container.getHost(),
            container.getMappedPort(33060),
            container.getDatabase()
            )

5     await ddl.createUserTable(
          await userRepo.getSession()
        )

6     dataUtils = new DataUtils(
            container.getUsername(),
            container.getUserPassword(),
            container.getHost(),
            container.getMappedPort(33060),
            container.getDatabase()
        )
    })

7   after(async ()=>{
8     await container.stop()
    })

9   await test('Container should be running', async (t)=>{
10    const queryResult =
          await container.executeQuery("SELECT 1 as res")
11    assert.ok(queryResult.includes("res\n1\n") )
    });

12  await test('Should create user', async(t)=>{
13    const name= faker.internet.username()
      console.log(`Test name: ${name}.`)
14    await userRepo.createUser({name: name})
15    const queryResult = await dataUtils.getUserByName(name)
16    assert.equal(1, queryResult.length)
17    assert.equal(name, queryResult[0].name)
    })

  })
```

Our Test File

```
import { test, describe, before, after } from 'node:test'
import { strict as assert } from 'node:assert'
import {MySqlContainer} from "@testcontainers/mysql"
import { faker } from '@faker-js/faker'
import { ddl } from "../setup/ddl.js"
import UserRepo from "../repository/userRepository.js"
import DataUtils from "../utils/DataUtils.js"

describe('Scott\'s Amazing Test Demo!!', async (t) => {
    let container
    let userRepo
    let dataUtils
```

Our Test File

```
2  before(async ()⇒{
3      container = await new MySqlContainer()
           .withExposedPorts(3306, 33060)
           .start();

4      userRepo = new UserRepo(
           container.getUsername(),
           container.getUserPassword(),
           container.getHost(),
           container.getMappedPort(33060),
           container.getDatabase()
           )

5      await ddl.createUserTable(
         await userRepo.getSession()
       )

6      dataUtils = new DataUtils(
           container.getUsername(),
           container.getUserPassword(),
           container.getHost(),
           container.getMappedPort(33060),
           container.getDatabase()
       )
   })
```


Our Test File

```
7 after(async ()⇒{
    8 await container.stop()
  })
```

Our Test File

```
9  await test('Container should be running', async (t)⇒{
10   const queryResult =
         await container.executeQuery("SELECT 1 as res")
11   assert.ok(queryResult.includes("res\n1\n") )
   });
```

Our Test File

```
12 await test('Should create user', async(t)⇒{
13     const name= faker.internet.username()
       console.log(`Test name: ${name}.`)
14     await userRepo.createUser({name: name})
15     const queryResult = await dataUtils.getUserByName(name)
16     assert.equal(1, queryResult.length)
17     assert.equal(name, queryResult[0].name)
   })


})
```

UserRepository.js

```
import * as mysqlx from '@mysql/xdevapi'
export default class UserRepo{
    #connectionUrl
1   #pool
2   constructor(dbUser, dbPassword, dbHost, dbPort, schemaName) {

3       this.#connectionUrl =
            `mysqlx://${dbUser}:${dbPassword}@${dbHost}:${dbPort}/${schemaName}`

4       this.#pool = mysqlx.getClient(this.#connectionUrl, {
            pooling: {
                enabled: true,
                maxSize: 10,
                maxIdleTime: 20000,
                queueTimeout: 5000
            }
        })
    }

    async getSession(){
5       return await this.#pool.getSession()
    }

6   async createUser(user) {
        const session = await this.getSession();
7       const db = session.getSchema();
        const table = db.getTable('user');
8       table.insert(['name'])
            .values(user.name)
            .execute();
9       session.close();
    }
}
```

UserRepository.js

```
import * as mysqlx from '@mysql/xdevapi'
export default class UserRepo{
    #connectionUrl
1   #pool
2   constructor(dbUser, dbPassword, dbHost, dbPort, schemaName) {

3       this.#connectionUrl =
            `mysqlx://${dbUser}:${dbPassword}@${dbHost}:${dbPort}/${schemaName}`

4       this.#pool = mysqlx.getClient(this.#connectionUrl, {
            pooling: {
                enabled: true,
                maxSize: 10,
                maxIdleTime: 20000,
                queueTimeout: 5000
            }
        })
    }
```

UserRepository.js

```
5   async getSession(){
        return await this.#pool.getSession()
    }
```

UserRepository.js

```
6   async createUser(user) {
        const session = await this.getSession();
7       const db = session.getSchema();
        const table = db.getTable('user');
8       table.insert(['name'])
                 .values(user.name)
                 .execute();
9       session.close();
    }
}
```

Helper Files

DataUtils.js

```
import * as mysqlx from '@mysql/xdevapi'
export default class DataUtils{
1   #connectionUrl
    #pool
2   constructor(dbUser, dbPassword, dbHost, dbPort, schemaName) {

3       this.#connectionUrl =
            `mysqlx://${dbUser}:${dbPassword}@${dbHost}:${dbPort}/${schemaName}`

4       this.#pool = mysqlx.getClient(this.#connectionUrl, {
            pooling: {
                enabled: true,
                maxSize: 10,
                maxIdleTime: 20000,
                queueTimeout: 5000
            }
        })
    }

5   async getSession(){
        return await this.#pool.getSession()
    }

6   async getUserByName(name) {
        let ret = [];
7       const session = await this.getSession();
8       const sql = `select id, name from user where name = '${name}'`
        const rows = await session.sql(sql).execute()
9       ret = this.formatData(rows)
        session.close();
        return ret;
    }

10  formatData(rows){
        const data = rows.toArray()
        const columns = rows.getColumns()
        let ret = [];
        data.forEach((row) =>{
            let obj = {};
            row[0].forEach((item,i)=>{
                obj[columns[i].getColumnLabel()] = item;
            })
            ret.push(obj)
        })
        return ret;
    }
}
```

Helper Files

DataUtils.js

```
import * as mysqlx from '@mysql/xdevapi'
export default class DataUtils{
    #connectionUrl
1   #pool
2   constructor(dbUser, dbPassword, dbHost, dbPort, schemaName) {

        3 this.#connectionUrl =
            `mysqlx://${dbUser}:${dbPassword}@${dbHost}:${dbPort}/${schemaName}`

        4 this.#pool = mysqlx.getClient(this.#connectionUrl, {
            pooling: {
                enabled: true,
                maxSize: 10,
                maxIdleTime: 20000,
                queueTimeout: 5000
            }
        })
    }
```

Helper Files

DataUtils.js

```
5  async getSession(){
       return await this.#pool.getSession()
   }
```

Helper Files

DataUtils.js

```
6  async getUserByName(name) {
       let ret = [];
7      const session = await this.getSession();
8      const sql = `select id, name from user where name = '${name}'`
       const rows = await session.sql(sql).execute()
9      ret = this.formatData(rows)
       session.close();
       return ret;
   }
```

Helper Files

DataUtils.js

```
formatData(rows){
    const data = rows.toArray()
    const columns = rows.getColumns()
    let ret = [];
    data.forEach((row) =>{
        let obj = {};
        row[0].forEach((item,i)=>{
            obj[columns[i].getColumnLabel()] = item;
        })
        ret.push(obj)
    })
    return ret;
  }
}
```

Running the Tests

```
 node --test 1
Test name: Rodrigo_Bernhard61. 2
▶ Scott's Amazing Test Demo!!
  ✔ Container should be running (64.001ms)
  ✔ Should create user (19.057708ms)
✔ Scott's Amazing Test Demo!! (11088.665792ms)
ℹ tests 2      3
ℹ suites 1     4
ℹ pass 2       5
ℹ fail 0
ℹ cancelled 0
ℹ skipped 0
ℹ todo 0
ℹ duration_ms 11535.621333 6
```

Best Practices

AVOID HARD-CODED VALUES!!

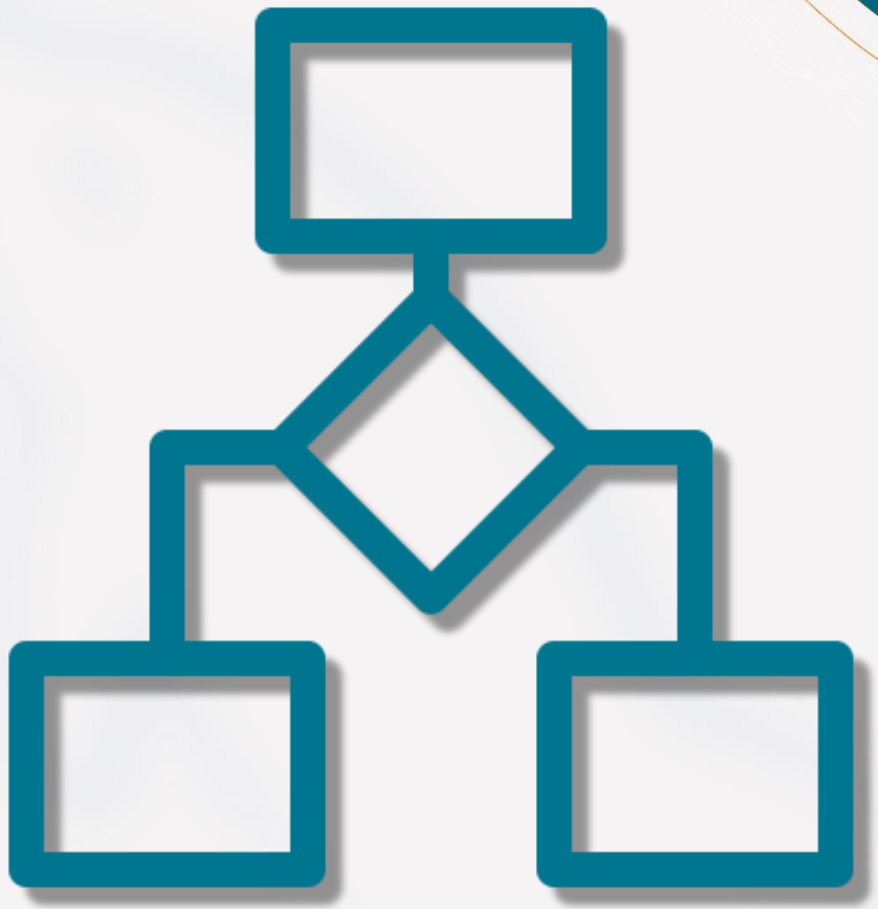
Best Practices

```
await test('My test', async() => {
  const testVal = myObj.addNumbers(4,6)
  assert.equal(testVal, 10)
})
```

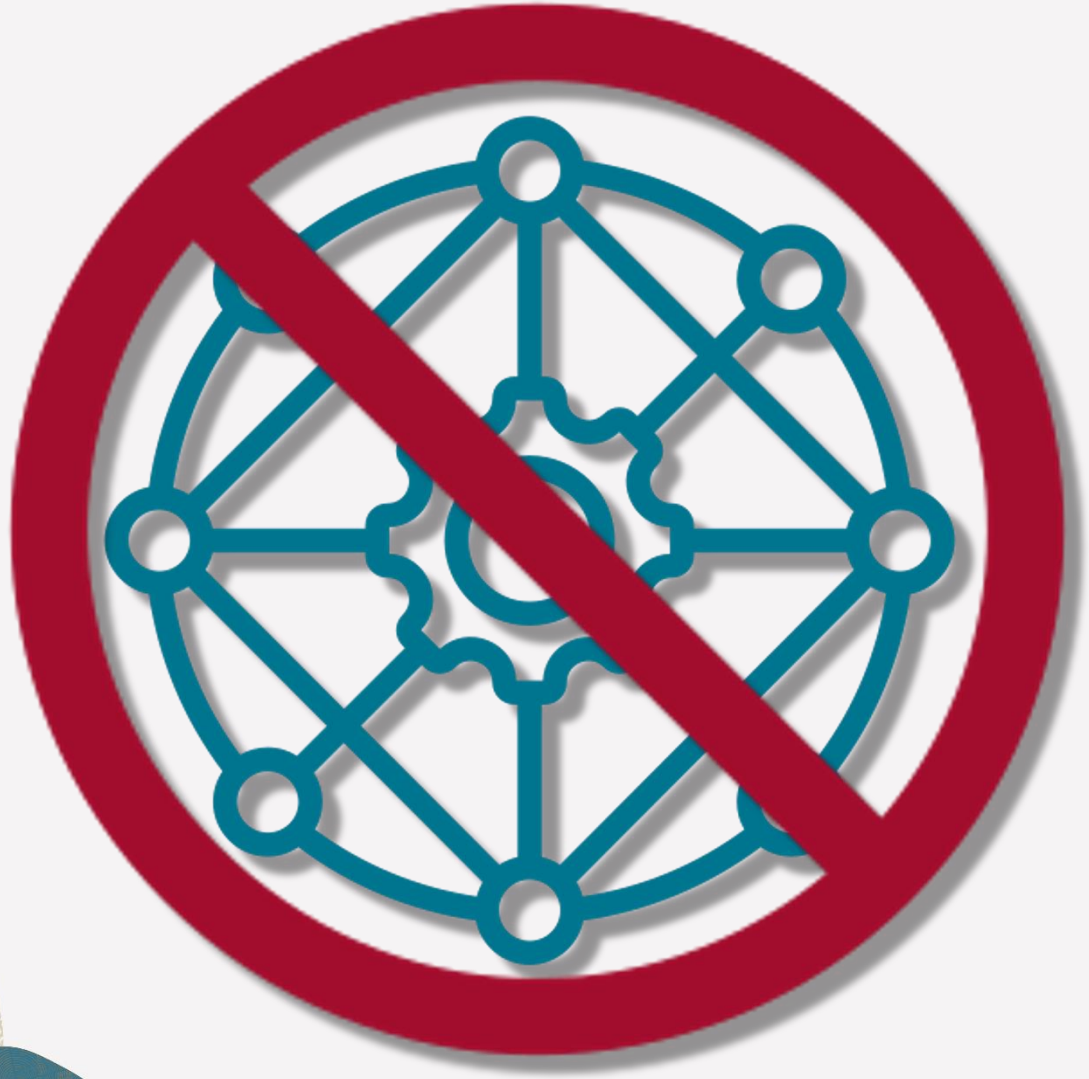
```
addNumbers(num1, num2){
  return 10
}
```

Best Practices

- Have a test for **EVERY** logical condition.
  - Every `if()` should have at least 2 tests.
  - Every `case` in a switch/case should have at least one test.
    - And one for the default case.



Best Practices



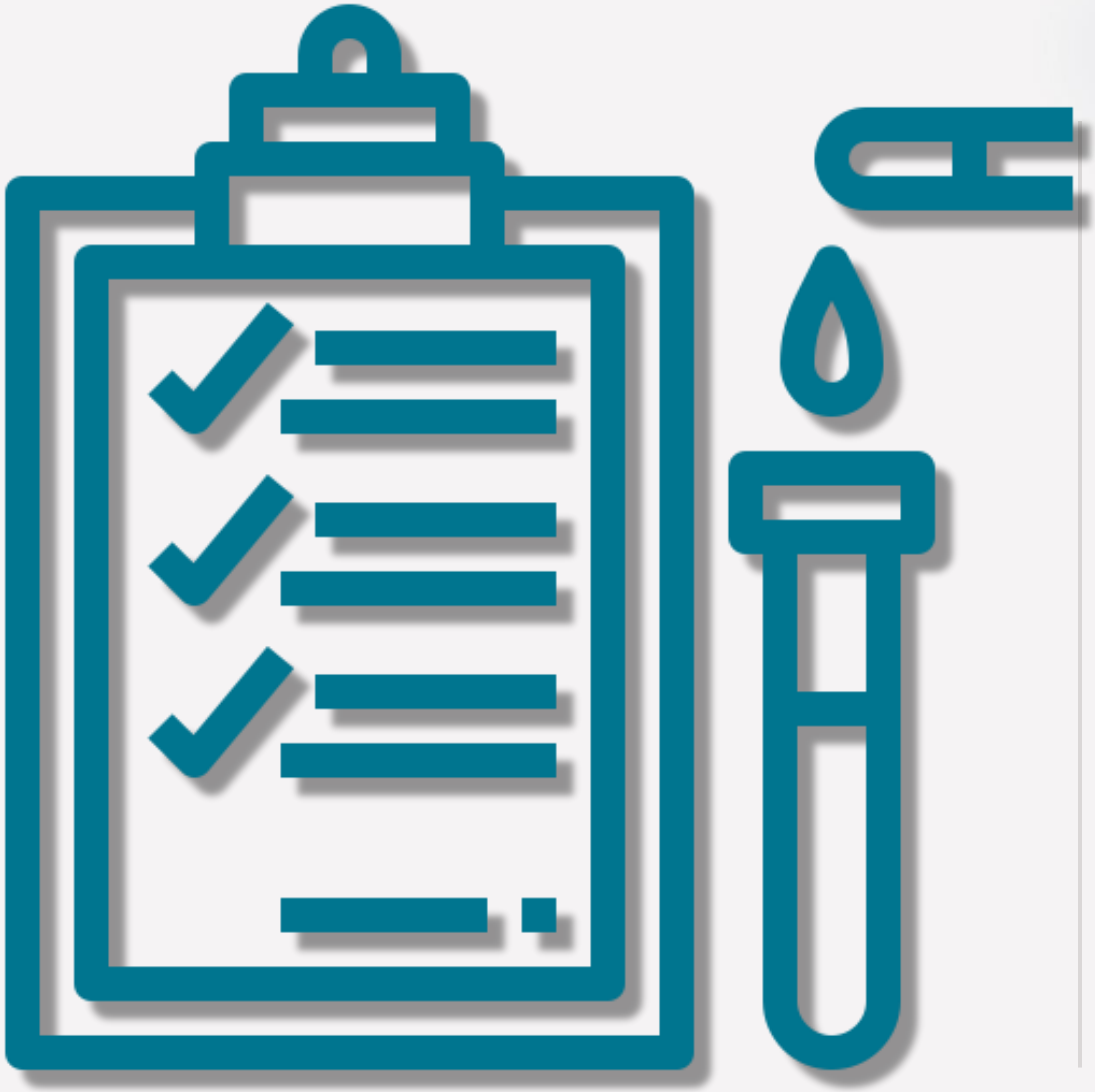
- Each unit test should only test **ONE** possible scenario
  - A single unit test should not check multiple logical conditions
- Unit tests should be simple
  - More complexity means more trouble
  - Makes code easier to read and follow

AVOID HARD-CODED VALUES!!

Best Practices

- Each Unit test should be independent of every other test
  - The success of a test should **NEVER** rely on the result of any other test
- Mock data and stub requests to external functions or APIs

Best Practices



- All tests should be automated!
  - Preferably part of your CI/CD process
- Use unit tests **AND** integration tests!

AVOID HARD-CODED VALUES!!

How to reach me…

- **Email:** scott.stroz@oracle.com
- **BlueSky:** @stroz.dev
- **Mastodon:** @sstroz
- **LinkedIn:** scott-stroz

Q&A

Thank You!