

Integration Testing using sbt, scalatest and Docker

Emanuele Blanco - @manub

Moneyfarm

we always want to build great software
or, at least, software that works...



unit testing

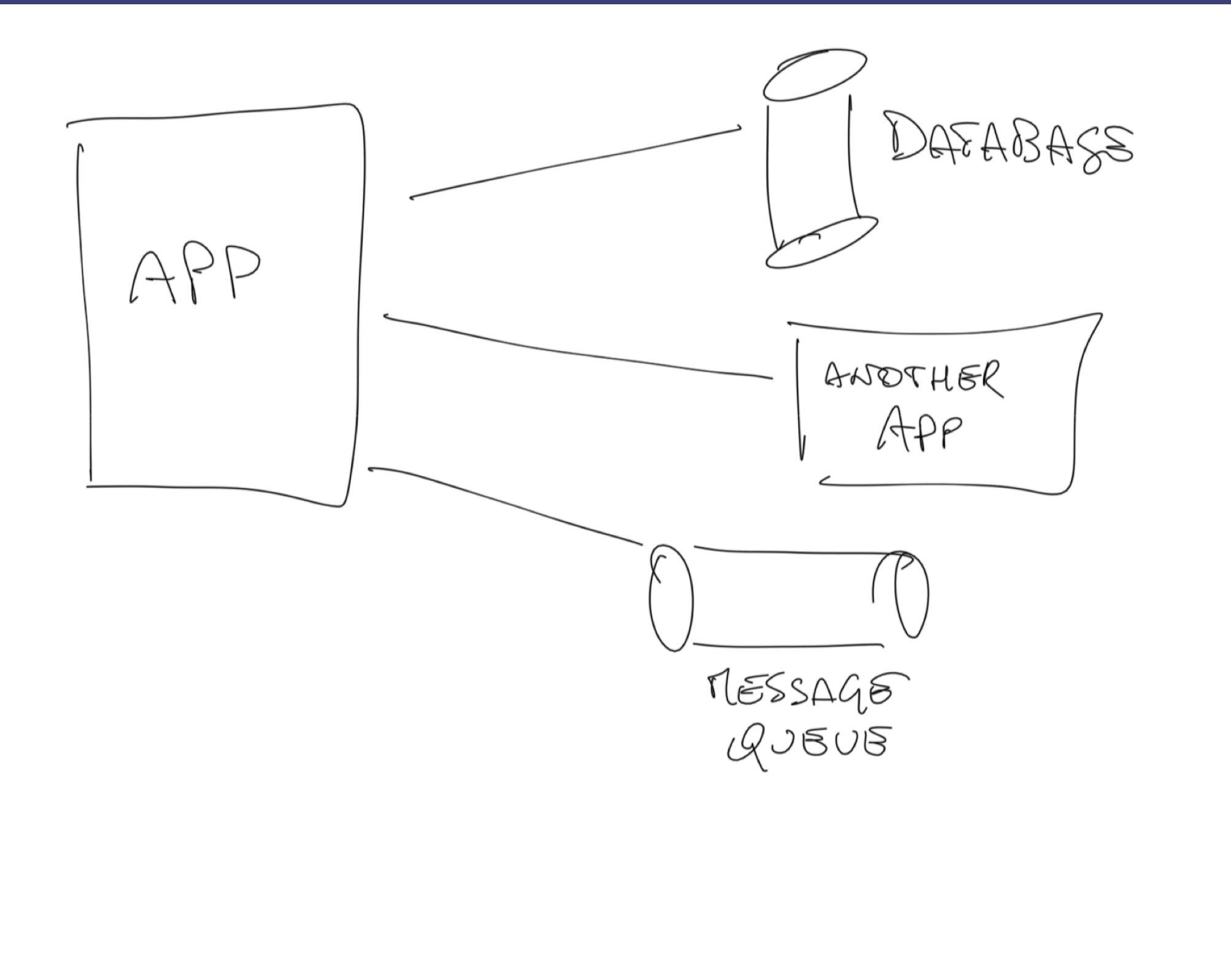
with Scalatest

```
case class Rectangle(width: Int, height: Int) {  
    lazy val area: Int = width * height  
}
```

```
"Rectangle" should {  
    "calculate correctly the area" in {  
        Rectangle(2, 3).area shouldBe 6  
    }  
}
```

sbt test runs the unit tests

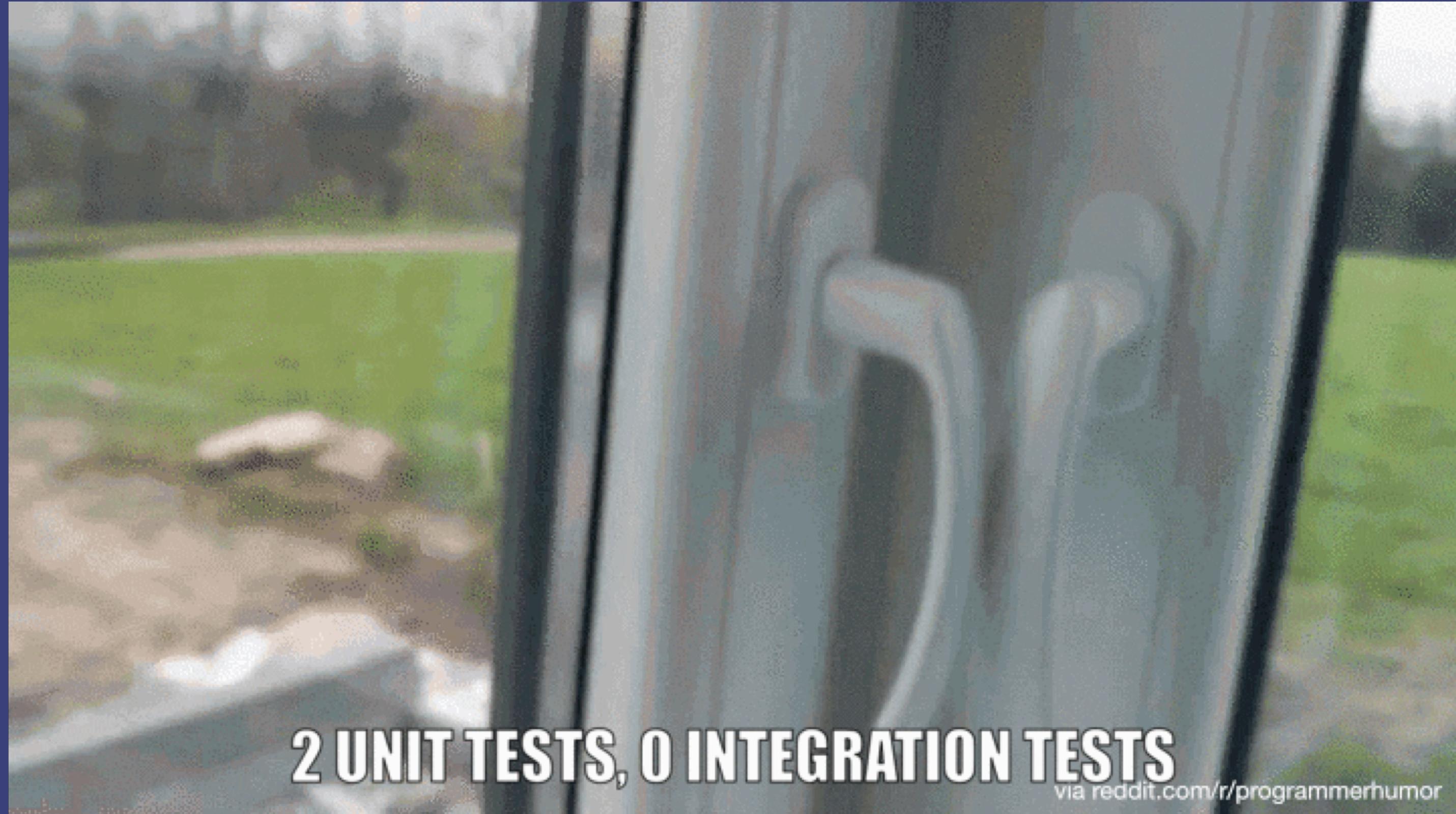
are unit tests enough?



```
case class User(username: String,  
                firstName: String,  
                lastName: String)  
  
class UserRepository {  
    // some database configuration...  
  
    def getAllUsers(): Task[List[User]] = ???  
}
```

**how can we test code that goes outside
the application boundaries?**

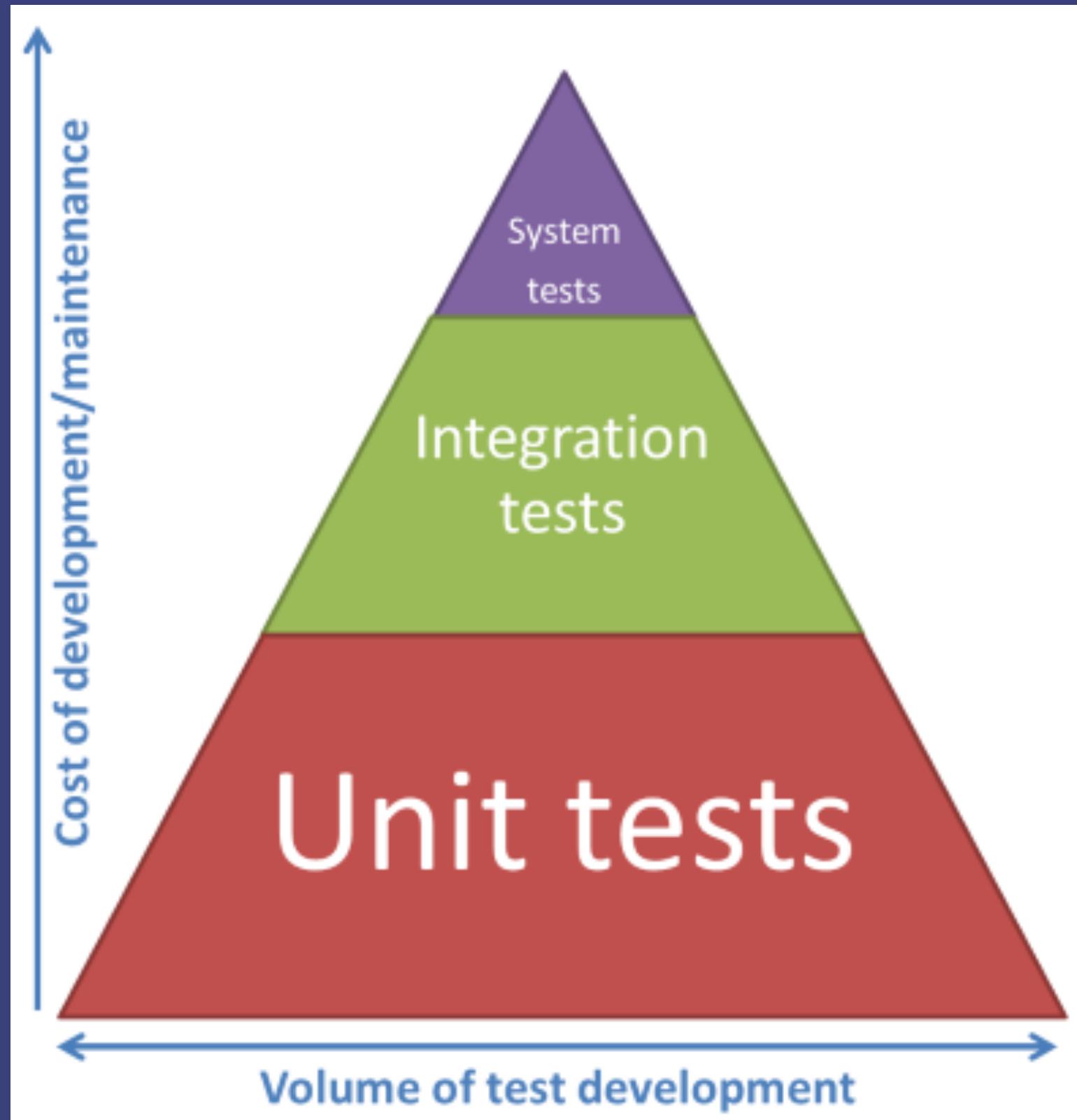
integration testing



via reddit.com/r/programmerhumor

integration tests

- make sure previously tested units work correctly with each other
- can go outside the application boundaries
 - they're usually slower



should be isolated from other tests in your build

sbt configuration

```
lazy val root = (project in file("."))  
.configs(IntegrationTest)  
.settings(commonSettings, Defaults.itSettings)
```

- the code will go in src/it
- run those tests using sbt it:test
- by default, parallel execution is disabled

**how to test
UserRepository?**

start Postgres Programmatically

if we have the sources of our dependency and it runs on the JVM, we could start it programmatically in our test

 [manub / scalatest-embedded-kafka](#)

[Unwatch](#) 17 [Star](#) 181 [Fork](#) 58

[Code](#) [Issues 8](#) [Pull requests 0](#) [Projects 1](#) [Wiki](#) [Insights](#) [Settings](#)

A library that provides an in-memory Kafka instance to run your ScalaTest specs against.

[Edit](#)

but Postgres doesn't run on the JVM 😞

H2

H2 is a in-memory database written in Java. It has a compatibility mode for Postgres and MySQL amongst others but:

- it's not like-for-like (see [here](#))
- SQL triggers are not supported
- in the end **is not the same database as the one we will use in production**

Use a Real Postgres instance

- could be shared with other clients
- requires additional resources (i.e. a server)

can we provision a Postgres instance on demand?



Docker

- runs applications (distributed as images) in containers
- the applications run in the same environment (the container), regardless of the host
- most vendors publish images of their software...
 - including postgres!! 

starting Postgres on Docker

```
docker run  
  --name my-postgres  
  -e POSTGRES_USER=user  
  -e POSTGRES_PASSWORD=mysecretpassword  
  -d  
  postgres:latest
```

starting Postgres on Docker for integration tests

two different approaches

- control container lifecycle outside the test (i.e. running the container before the `it:test` phase)
- write a docker integration within your tests, to control the container lifecycle

docker-it-scala

<https://github.com/whisklabs/docker-it-scala>

- allows you to use Docker Java or the Spotify client to connect to Docker
- supports both Scalatest and Specs2

```
libraryDependencies += Seq(  
  "com.whisk" %% "docker-testkit-scalatest" % "0.9.5" % IntegrationTest,  
  "com.whisk" %% "docker-testkit-impl-spotify" % "0.9.5" % IntegrationTest  
)
```

```
trait DockerKit {  
    implicit def dockerFactory: DockerFactory  
  
    def dockerContainers: List[DockerContainer] = Nil  
  
    // other values  
}  
  
trait DockerKitSpotify extends DockerKit { ... }
```

```
trait DockerPostgresService extends DockerKit with PostgresConfiguration {

  import scala.concurrent.duration._

  val PostgresAdvertisedPort = 5432

  lazy val postgresContainer: DockerContainer =
    DockerContainer("postgres:latest")
      .withPorts(PostgresAdvertisedPort -> Some(postgresPort))
      .withEnv(s"POSTGRES_USER=$postgresUsername",
              s"POSTGRES_PASSWORD=$postgresPassword",
              s"POSTGRES_DB=$postgresDatabase")
      .withReadyChecker(
        new PostgresReadyChecker(postgresUrl,
                                 postgresUsername,
                                 postgresPassword).looped(15, 1.second)
      )

  abstract override def dockerContainers: List[DockerContainer] =
    postgresContainer :: super.dockerContainers

}
```

```
class PostgresReadyChecker(url: String, username: String, password: String)
  extends DockerReadyChecker {

  override def apply(container: DockerContainerState)(
    implicit docker: DockerCommandExecutor,
    ec: ExecutionContext): Future[Boolean] =
    container
      .isRunning()
      .map(_ =>
        Try {
          DriverManager
            .getConnection(url, username, password)
            .close()
          true
        }.getOrElse(false))
}
```

```
class UserRepositorySpec extends WordSpec
  with Matchers // plus other scalatest goodies
  with DockerTestKit // scalatest support
  with DockerKitSpotify // docker client implementation
  with DockerPostgresService { // define the container

    val repository = new UserRepository

    "UserRepository" should {
      "get all the users" in {

        val user = User("manub", "Emanuele", "Blanco")

        insertIntoDb(user)

        val retrievedUsers = repository.getAllUsers()

        retrievedUsers.unsafeRunAsyncFuture().futureValue shouldBe List(user)
      }
    }

    private def insertIntoDb(user: User) = {
      // inserts a user into the db, but not using the class under test
    }
}
```

code sample (and slides) available at:
<https://github.com/manub/it-testing>

thanks!