Roles: a viable alternative to Microservices and Monoliths

Septimal Mind Ltd

team@7mind.io

July 25, 2018

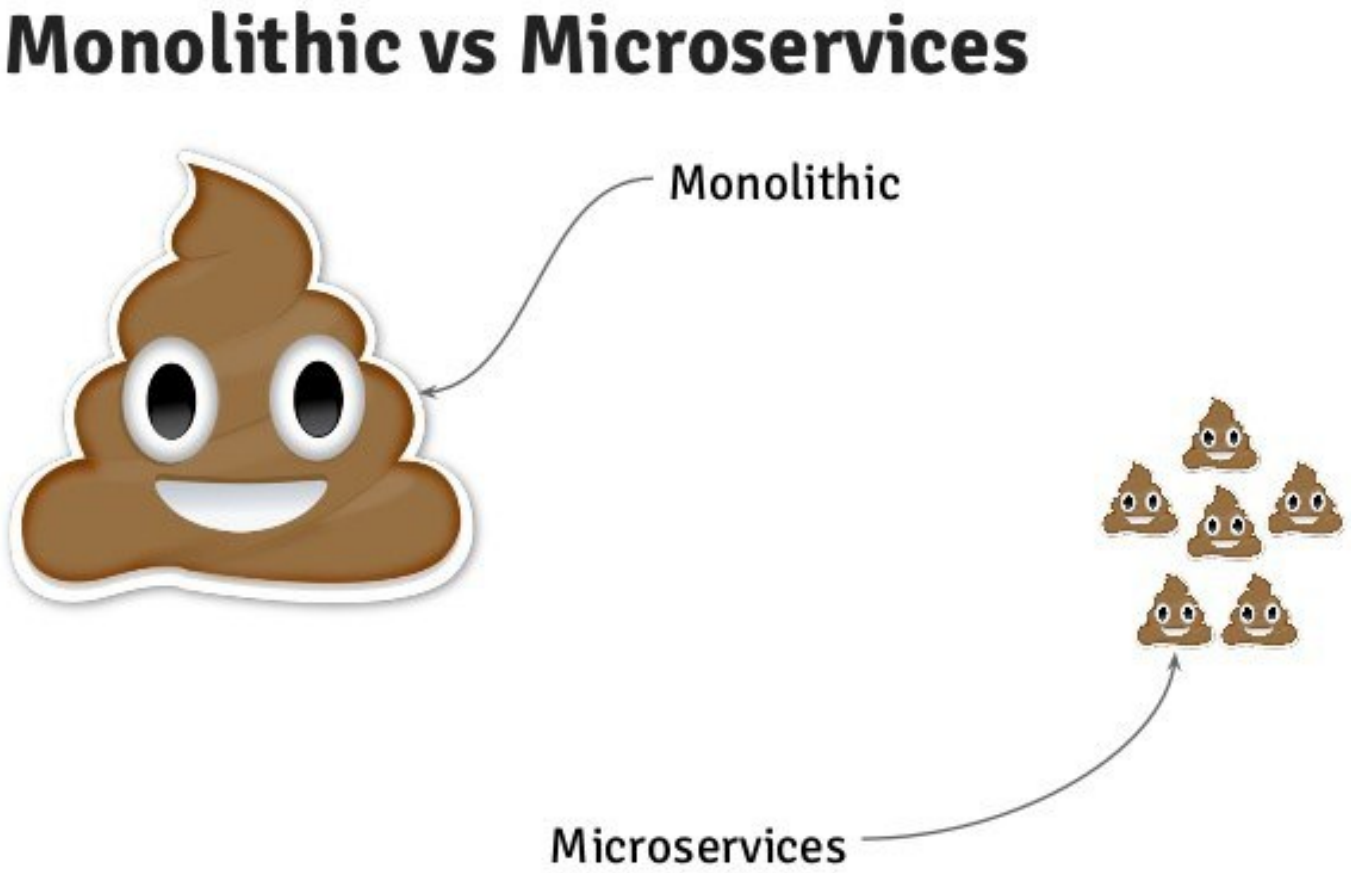
Monoliths are broken

We all know that Monoliths aren't nice:

- Provoke tight-coupling and non-SOLID code,
- When you deploy you need to deploy everything (not always a bad thing),
- Startup time is usually high,
- Inconvenient when your codebase is large: you may need to rebuild to much, etc,
- Dependency convergence.

Structuring, conventions and different enforceable boundaries may mitigate some issues.
Pulling conservative parts out of the main codebase into independent libraries helps.

You've seen this before



The alternative – Microservices – is just sick

- Evil Dependencies and neccessary *Shared State*,
- Deployment and maintenance becomes more complicated with each new dependency,
- Inevitably Distributed: all *fundamental* problems are in place. CAP theorem?,
- Cascading failures,
- Accidental Incompatibilities: you may notice them too late. Amplified by cascade failures,
- Computing Density: 50 JVM microservices in Docker on a single machine? Really?,
- Overcomplicated Development Flows: a simple [integration[1]] test requires many things done in right order,
- Time, interference and reproducible builds: isolated environments are expensive to setup.

[1] Not a constructive definition. Subject of another presentation

Observations on Microservices

- *Software components* are still in place – in form of Services. All the potential pitfails of component design apply to microservices,
- Many checks which may be done by your compiler in case of monolithic design have to be performed by integration test suites (in case you have them, it's hard to write),
- Many tasks are delegated to operations: in case of a monolith your dependency graph is being processed by your DI framework[1], in case of microservices it is. . . in worst case processed by people, in best case requires an orchestration tool. But we don't have any single great tool yet,
- An orchestration tool does *exactly the same job* your DI framework may do. And lot more things.

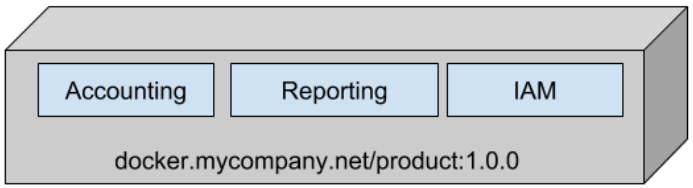
[1] Or an alternative mechanism with the same purpose

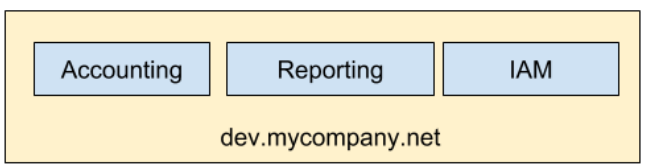
Roles: the idea to rescue

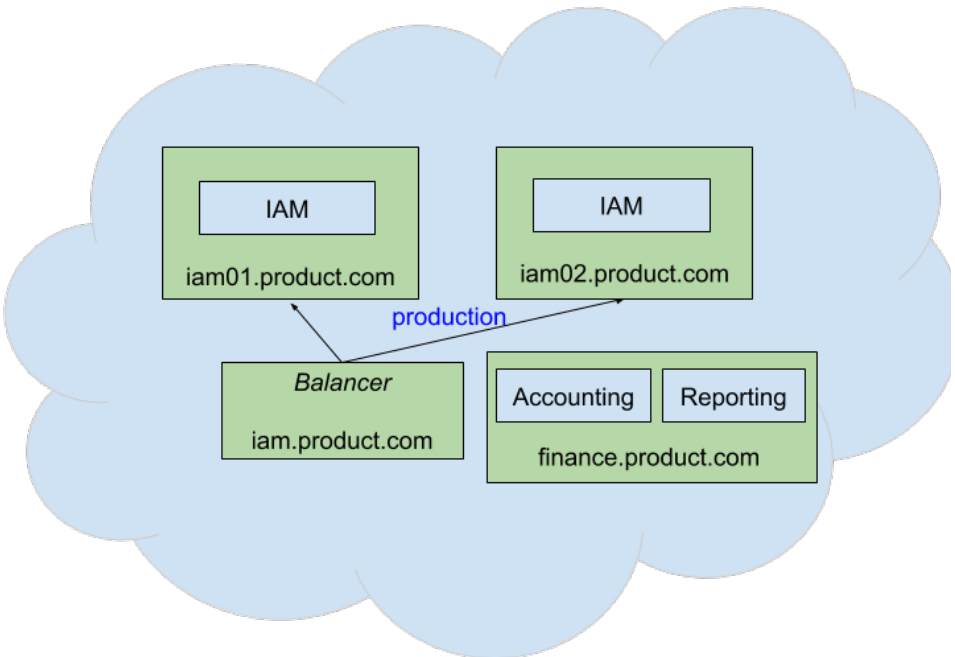
We need something with all the positive traits of Microservices and Monoliths but without negative.

- Let's run multiple services. . . in one service. Well, one process,
- Let's choose which *roles* we wish to assign to a before we start it,
- Roles come from Classpath, may be discovered dynamically or statically,
- It resembles Containers and OSGi,
- Containers have some disadvantages. They provide you a lot so we get used to think that they are sluggish on startup. Isolated classloaders are inconvenient. Dynamic DI is a nightmare,
- Let's get rid of Dynamic DI. And let's keep isolation optional.

Role-based approach







Benefits

- Higher Density makes Development Flows better: instant startup of any combination of roles,
- Higher Density makes Operations easier: any combination of roles for any container. You don't need to start 5 containers to have 5 services with low load profile,
- Smaller distribution size: Just one image, less traffic,
- More static checks ⇒ higher reliability,
- Easier to setup environments (just run a process) ⇒ easier to perform integration testing ⇒ higher reliability, quicker delivery,
- Quicker builds.

Missing things and potential pitfails

- Startup time. Do you remember Tomcat? Design matters[1],
- Heterogenous systems: you can't fit C#, Scala and Go into a single process. Though you still can greatly improve heterogenous systems by using roles,
- Dependency convergence. In case you want roles to be completely independent you need isolated classloaders. Solved in OSGi. Expensive, inconvenient,
- Distributed communication between Roles: most likely you don't want it. Otherwise you need a mechanism similar to: distrubuted DI framework built around *Service Discovery* and *Cluster State* concepts. See: `dOSGi`,
- You need to build a nice CD pipeline.

[1] We have something to add to well-known recepies. Subject of another presentation

DISTAGE

Next-gen Dependency Injection Framework for Scala

Generative, Modular, R✪les and Garbage Collection included

Quick overview

- Next generation of DI.
- Generative, built on PPER principle. Allows you to plan context provisioning, edit the plan, then execute it,
- Fully aware of Scala typesystem, allows you to fuse FP and OOP lot better than before,
- Bundled Garbage Collector,
- *Bundled Roles mechanism* built as an extension. Cheap for developer because of Garbage Collection,
- Bundled `typesafe-config` support built as an extension,
- Kind of. . . a new paradigm. We are still discovering patterns and possibilities. Subject of separate presentation.

How does it look like[1]?

```scala
@RoleId("testservice")
class TestService(httpServer: HttpServer) extends IzService {
  override def start(): Unit = httpServer.start()
  override def stop(): Unit = httpServer.stop()
}
class TestPlugin extends PluginDef {
  make[IzService].named("testservice").from[TestService]
}
object TestLauncher {
  // you may run it like test.jar test-service other-service
  def main(args: Array[String]): Unit = {
    new IzRoleApp().main(args)
  }
}
```

[1] Not published on GitHub yet, will be available soon. More details to follow.

```
➜ ~ docker run -ti --rm -e "JAVA_ARGS_TAIL=-Djava.awt.headless=false" [redacted].jfrog.io/develop/tg-launcher-app:v1.0.0 -ds role -i bookkeeper
Working in  : /app
OOM Command : /bin/sh /app/oom-killer.sh %p
App Args    : net.[redacted].tg.launcher.TGLauncher  -ds role -i bookkeeper
-> External environment NOT defined: tg-launcher-app.env
-> External classpath file NOT defined: tg-launcher-app.classpath
-> Default GC details parameters are in use
-> Default GC parameters are in use
-> Default heap parameters are in use
-> Default jvm parameters are in use
Command:
exec java -d64 -server -noverify -cp "lib/*" -Xms256m -Xmx2048m -Xss1m -XX:ReservedCodeCacheSize=100m -XX:MaxMetaspaceSize=256m -XX:+UseConcMarkSweepGC -XX:+UseParNewGC -XX:+DoEscapeAnalysis -verbose:gc -XX:+PrintTenuringDistribution -XX:+PrintGCDetails -XX:+PrintGCDateStamps -XX:+PrintGCTimeStamps -XX:+PrintGCApplicationStoppedTime -XX:+UseGCLogFileRotation -XX:NumberOfGCLogFiles=10 -XX:GCLogFileSize=1M -XX:+HeapDumpOnOutOfMemoryError -XX:OnOutOfMemoryError="/bin/sh /app/oom-killer.sh %p" -XX:HeapDumpPath=/var/log/heapdump/tg-launcher-app -Xloggc:/var/log/gc/tg-launcher-app/gc.log -Djava.security.egd=/dev/./urandom -Djava.net.preferIPv4Stack=true -Dapplication.id=tg-launcher-app -Djava.awt.headless=false -Dcom.sun.management.jmxremote.port=5000 -Dcom.sun.management.jmxremote -Dcom.sun.management.jmxremote.authenticate=false -Djmxmp.listen=true -javaagent:jolokia-jvm-agent.jar=port=5001,host=*,user=monitoring,password=monitoring net.[redacted].tg.launcher.TGLauncher -ds role -i bookkeeper
I> No access restrictor found, access to any MBean is allowed
Jolokia: Agent started with URL http://172.17.0.3:5001/jolokia/
I 2018-07-24T21:01:04.291Z[GMT]  main:1  (TGAppBootstrapStrategy.scala:182) msg=Application is about to start : details=0.2.0 @ f8c2dd3481d8450490fedacf904820ab83dbc015#f8c2dd3481d8450490fedacf904820ab83dbc015*, net.[redacted]:tg-launcher, 2018-07-19T22:32:56.297Z[UTC] (jdk: 1.8.0_111, by: root)
I 2018-07-24T21:01:04.416Z[GMT]  main:1  (TGAppBootstrapStrategy.scala:182) msg=... using tg-unified-sdk : details=0.2.0 @ f8c2dd3481d8450490fedacf904820ab83dbc015#f8c2dd3481d8450490fedacf904820ab83dbc015*, net.[redacted]:tg-launcher, 2018-07-19T22:32:56.297Z[UTC] (jdk: 1.8.0_111, by: root)
I 2018-07-24T21:01:04.430Z[GMT]  main:1  (TGAppBootstrapStrategy.scala:182) msg=... using izumi-r2 : details=0.5.37 @ aa22a2766584b27c040d47a2b0379c165f8ecb1a#aa22a2766584b27c040d47a2b0379c165f8ecb1a*, com.github.pshirshov.izumi.r2:distage-app, 2018-07-19T18:43:23.274Z[UTC] (jdk: 1.8.0_172, by: liyon)
I 2018-07-24T21:01:10.116Z[GMT]  main:1  (TGAppBootstrapStrategy.scala:117) Available app plugins=33 and bootstrap plugins=0, app bindings=153available ...
I 2018-07-24T21:01:10.251Z[GMT]  main:1  (TGAppBootstrapStrategy.scala:138) Available roles=
 - adhawk, net.[redacted].tg.adhawk.AdhawkRole, source=1.0.0 @ 7d979b79a12a43c902977b3cc949c50116246092#7d979b79a12a43c902977b3cc949c50116246092*, net.[redacted]:adhawk-role, 2018-07-19T23:02:25.712Z[UTC] (jdk: 1.8.0_111, by: root)
 - bookkeeper, net.[redacted].tg.bookkeeper.BookkeeperRole[cats.effect.IO,net.[redacted].security.RequestContext], source=1.0.0 @ 647501d7b91bb1f90b003ba280e65618b64c7eff#647501d7b91bb1f90b003ba280e65618b64c7eff*, net.[redacted]:tg-bookkeeper, 2018-07-19T23:04:08.856Z[UTC] (jdk: 1.8.0_111, by: root)
 - configwriter, net.[redacted].tg.launcher.ConfigWriter, source=0.2.0 @ f8c2dd3481d8450490fedacf904820ab83dbc015#f8c2dd3481d8450490fedacf904820ab83dbc015*, net.[redacted]:tg-launcher, 2018-07-19T22:32:56.297Z[UTC] (jdk: 1.8.0_111, by: root)
 - jobot, net.[redacted].tg.adhawk.JobotRole, source=1.0.0 @ 7d979b79a12a43c902977b3cc949c50116246092#7d979b79a12a43c902977b3cc949c50116246092*, net.[redacted]:adhawk-role, 2018-07-19T23:02:25.712Z[UTC] (jdk: 1.8.0_111, by: root)
I 2018-07-24T21:01:10.255Z[GMT]  main:1  (TGAppBootstrapStrategy.scala:139) Requested roles=
 - bookkeeper
I 2018-07-24T21:01:10.262Z[GMT]  main:1  (TGAppBootstrapStrategy.scala:123) Disabled tags=
 - adhawk
 - configwriter
 - jobot
 - storage-production
I 2018-07-24T21:01:10.480Z[GMT]  main:1  (TGAppBootstrapStrategy.scala:91) Loaded app bindings=93 and bootstrap bindings=0...
I 2018-07-24T21:01:13.304Z[GMT]  main:1  (TGStarter.scala:21) services=1; tasks=0; components=3 are going to start...
I 2018-07-24T21:01:13.309Z[GMT]  main:1  (TGStarter.scala:25) Starting component service=net.[redacted].cassandra.DummyStorageContext@d611f1c...
I 2018-07-24T21:01:13.311Z[GMT]  main:1  (StorageContext.scala:10) Initializing storage context; @type=const
I 2018-07-24T21:01:13.315Z[GMT]  main:1  (StorageContext.scala:29) Dummy storage context started; @type=const
I 2018-07-24T21:01:13.316Z[GMT]  main:1  (StorageContext.scala:12) Initializing storage context finished; @type=const
I 2018-07-24T21:01:13.319Z[GMT]  main:1  (TGStarter.scala:25) Starting component service=net.[redacted]tg.http.TgHttpComponent@51fc862e...
I 2018-07-24T21:01:13.326Z[GMT]  main:1  (TGStarter.scala:25) Starting component service=net.[redacted].tg.bookkeeper.BookkeeperRole@28e0e464...
I 2018-07-24T21:01:13.330Z[GMT]  Thread-6:67  (HttpServerLauncher.scala:23) Going to expose size=8:endpoints IDL endpoints with HTTP...
I 2018-07-24T21:01:13.329Z[GMT]  main:1  (BookkeeperRole.scala:16) BookkeeperRole service started!
I 2018-07-24T21:01:13.333Z[GMT]  main:1  (TGStarter.scala:32) Startup finished, joining on main thread...; @type=const
I 2018-07-24T21:01:13.339Z[GMT]  Thread-6:67  (Http4sServer.scala:36) Starting HTTP server...; @type=const
I 2018-07-24T21:01:13.444Z[GMT]  Thread-6:67  (Http4sServer.scala:38) Http Server started on: interface=http://0.0.0.0:8080; mounted services=2
```

Status and things to do

Distage is:

- ready to use,
- in real production for 4 months.

Our plans:

- Make Roles opensource. ASAP,
- Implement a Testkit,
- Support optional isolated classloaders (in foreseeable future),
- Check our GitHub: https://github.com/pshirshov/izumi-r2.

DIStage is just a part of our stack

We have a vision backed by our tools:

- Idealingua: transport and codec agnostic gRPC alternative with rich modeling language,
- LogStage: zero-cost logging framework,
- *Fusional Programming and Design* guidelines. We love both FP and OOP,
- *Continous Delivery* guidelines for Role-based process,
- *Percept-Plan-Execute* Generative Programming approach, abstract machine and computational model. Addresses Project Planning (see Operations Research). Examples: orchestration, build systems.

Altogether these things already allowed us to significantly reduce development costs and delivery time for our client.

More slides to follow.

Thank you for your attention

https://izumi.7mind.io/distage/

We're looking for clients, contributors, adopters and colleagues ;)

About the author:

- coding for 18 years, 10 years of hands-on commercial engineering experience,
- has been leading a cluster orchestration team in Yandex, "the Russian Google",
- implemented "*Interstellar Spaceship*" – an orchestration solution to manage 50K+ physical machines across 6 datacenters,
- Owns an Irish R&D company, https://7mind.io,
- Contacts: team@7mind.io,
- Github: https://github.com/pshirshov