Full Microservice Integration and Complete Cloud Capacities

by Philippe Preval

Introduction

Our broadly used Tango software, is an implementation of a microservice architecture. Tango was not initially designed to be “microservice”, it was more properly designed to implement a transactional, mission critical, Service Oriented Architecture “that works” (SOATW!). By “that works” I mean, performant, scalable, avoiding contentions and anarchic leeway and sustainable. It is clear that to make it sustainable, a SOA system must include major concepts: a data bus or universal messaging layer to anarchy (meaning the ability to define as many interfaces than relations between services) and a load-balancer in order to avoid contentions or stress points. Some state that these 2 concepts are the key differences between SOA and microservice. However, the definition of microservice is not so clear and sometimes it is nothing more than: “it is not a monolithic architecture”. So, the first thing we will do is define it, then we will outline what applies and what does not apply in Tango and finally we will present Tango v8.

What is a microservice architecture?

A microservice application is a collection of autonomous services, each of them doing one thing well, and when combined, work together to provide a global service. Instead of a single complex system (monolithic architecture), the aim is to build and manage a set of relatively simple services that might interact in complex ways. These services collaborate with each other through a messaging protocol. The idea is quite simple. Having a collection of little ships instead of a huge one. That metaphor is not totally wrong. Lots of little ships are easy to maneuver, if one is delayed the others can progress. However, you can quickly cover more space with your multiple ships and if one is sunk (bad feature, bad design…) the others can still fight. Of course, there are some intrinsic difficulties, first a light fleet requires more coordination, second it is not as easy to make it a robust battleship.

Anyway, microservices promise a better way to sustainably deliver business impact. Rather than a single monolithic unit, applications built using microservices are made up of loosely coupled, autonomous services. Building services that do one thing well, avoids the inertia and entropy of large applications.

Properties of microservices are:

- A single microservice should be highly cohesive: it should be responsible for some single capability within an application. Likewise, the less that each service knows about the inner workings of other services, the easier it is to make changes to one service — or capability — without forcing changes to others.
- A microservice owns its data store, if it has one. This reduces coupling between services because other services can only access data they don't own through the interface that a service provides.
- Microservices themselves, not the messaging mechanism that connects them nor another piece of software, are responsible for “choreography” and collaboration — the sequencing of messages and actions to perform some useful activity.
- Each microservice can be deployed independently. Without this, a microservice application would still be monolithic at the point of deployment.
- A microservice is replaceable. Having a single capability places a natural boundary on size; likewise, it makes the individual responsibility, or role, of a service easy to comprehend.

An exception to our design, are the concepts (load balancer and Databus) that are introduced above, which are not accepted by all publications, however, the differences with SOA are more shades than key points. For instance, microservices are responsible for coordinating actions in a system while this can be external in SOA (complex orchestration can be externalized). Others say that Microservices design is driven by business and SOA design is driven by technique (technical layers…). This is an expert's debate.

Microservices applications scale by:

- Adding instances of microservices. This is of course standard with microservices.
- Deploying multiple, identical instances of the application (like Monolithic applications. On this point there is no specific advantage for Microservices applications).
- Horizontal data partitions (ex: partitioning on account numbers). In theory this can be used both for microservices and monolithic applications. In practicum this is much more efficient with microservices.

So, a microservices application can scale along the 3 axes, as they have been defined by Abbott and Fisher in The Art of Scalability.

Five architectural principles structure microservices developments:

- Autonomy: each service operates and changes independently. They are loosely coupled and can be deployed independently (Services can be developed in parallel, by different teams…).
- Resilience: microservices is a mechanism for isolating failure.

- Transparency: system must be transparent as a global service is provided by multiple. microservices, in case of malfunction, it is imperative to know what fails and why.
- Automation: as a global service is provided by multiple microservices it is important to integrate deployment practices at very early stage.
- Alignment. aligning teams and developments around business concepts.

As mentioned above there are specific challenges and risks:

- The risk of dilution with too many entities (i.e. “nano-services”) providing nothing.
- The risk of having contracts and boundaries defined “one to one” with no global view.
  - The combination of these 2 risks leads to a baroque or mannerist architecture where you lose your way.
- The risk of having data everywhere and in every format.
- The risk of having developments with no vision or care of performance, security, or consistency.

The architecture demands management and vision.

Tango. What we implement, what we do not.

First, we at Lusis produce mission critical systems or applications that can't fail. They must never lose a single transaction, never lose an order, never tolerate an outage, etc… This is not the standard world of the “web apps”. This is such an important difference that only those who are in the mission critical business can really understand it, because failure of these mission critical systems could be the deciding factor between a managing director keeping their job or not.

Second, we are not realizing apps, projects or custom development. We are designing and developing a software that has to be economically competitive. This means for instance, that all development must look like the others, be written in the same way, with the same style, as we can't afford “specialist developer” for this work.t. This is close to the CBSD model of development that I won't expound upon here but to say it is fundamental in our development approach.

Third, we provide an IT infrastructure that will last for many years into the future. So, it must be designed to live and scale, change, and mutate for 10, to 20 years or more. The infinity of an IT time scale.

Therefore, we have some very specific requirements:

- We are responsible and accountable for the performance and high availability of the system. With that in mind, we cannot delegate the data transport to a third party application. We will not put ourselves in a position to blame a third party. Therefore, we must be responsible for all aspects the system.
- Microservices development teams can't negotiate the contracts and the boundaries. This is done by the Tango platform, the Tango API to access the Databus, the Tango Databus, and the Tango data dictionary. Of course, this is evolving and changing but at a different pace and bearing the global aim in mind.
- A microservice does not own its data store for many reasons.

  - First because we want to use the data and it is easier if the data are all in the same place.
  - Second because the data are more secure and encrypted.
  - Third because we export them to client's data lakes or storage. .
- We are performance driven, so we don't multiplicate the swapping between processes for the pleasure of beauty.

Having said that, a Tango application is clearly a collection of autonomous services, each of them doing one (or a few more) thing(s) well, that work together to provide a global service. They can be developed separately, and deployed and run independently. A Tango application scales properly along the 3 axes that were mentioned above.

As we are providing software in a restricted number of business spaces these services can be grouped in “families of services” that are doing the same kind of things at a conceptual level. It is important because each “family” or sub-family, have its own technique and set of libraries, tools, and ways to improve productivity. For instance, in a payment system, services types are:

- Cash recycling Exchanging data with payment networks: roughly converting external messages in the Tango bus and vice versa. Of course, a bit more complicated. We can imagine that dialing with Visa is in se not so far from dialoging with UPI (China) even if details are different. So, all the connections to payment networks (Visa, Mastercard, UPI, JCB, …), all the message connections with HOST systems (DDA & so on) are in the same family
- Managing devices: requires dealing with a message protocol but at the other end you have a terminal with attributes, counters, local data, stop/start commands, etc. Having said that an ATM is an ATM, a POS device is a POS device.
- Web services and APIs: this is similar to a messaging network like VISA or Mastercard at a conceptual level.
- Business services: making decisions (authorize a transaction or not), detecting fraud and orchestrating complex multi-leg transactions. Business services are of course agnostic to any external data format as it is normalized in the Tango data bus.
- Technical services: transporting, routing, logging and managing high availability. One of the key technical services is the Tango dispatcher that is in charge of technical routing and load-balancing.

The Tango application is a collection of autonomous services that are a combination of interfaces with payment networks, business services, technical services, etc. All of them exchanging data via the set of APIs of the data bus and the dispatcher.

The studying of the Susan Fowler's book Production-Ready Microservices was very useful to us as it allowed us to review and audit our Tango architecture from the criteria for Production Readiness that she defines. The review concluded that a strong majority of these criteria were met (95%). When we were observing that Tango was not matching one of those criteria it was either by mistake or by a decision. In that case, it was worthwhile to evaluate if this decision was still valid. For instance, we were not interested in implementing the capacity for a Tango application to auto-create instances of services if the

software was indicating there were not enough of them. For this reason: as we could not push the wall, or create CPU unit, it was useless or even negative to create new instances that would have further disturbed the machine and worsened the situation. Of course, this is no longer true with the Cloud capacities.

From this review we deducted a list of Tango architecture features that constituted the base of the Tango Version 8 roadmap.

Tango Version 8

We will limit the presentation to the two most important changes that are creating a true disruption considering Cloud capacity.

Reviewing the "dispatcher"

As mentioned above the Tango dispatcher is in charge of the technical routing and load-balancing. This is a very robust, very efficient (micro seconds to process), multi-instanced service. However, mixing these two functionalities has some drawbacks: the routing context is in the dispatcher (timeout, multi-step rule, …), the dispatcher uses Tango events like a normal Tango service and therefore ,the dispatcher is not context free so can't be easily "cloudified".

The dispatcher will be split into two parts:

- A routing library that contains all the message routing, timeout handling, failure handling code. This will be embedded in each routing consumer service.
- A message router that allow routing messages between processes based of data provided in the messages itself.

When a Tango service sends an event (either a request or a notification), the destination is computed by applying the routing rules on the message. There are two cases:

- The service is local (in the same process).
- The service is not local (in another process).

When the destination service is local, the event is directly sent to the service using direct COM -a Tango process that knows each service composing it and can push directly the events inside a specific instance of a specific service), otherwise the message will be sent to the message router.

The message router will not be a "Tango service", it will use a lightweight Tango process and exchange messages using an optimized binary protocol not using standard Tango messaging.

A typical event routing message will contain information about the sender, the target and the content. The message router will be completely agnostic about the message content, it will only use the routing data for delivery.

The message router gathers the list of connected services (all the services are connecting to all message

routers and keep the connection alive like is done with the dispatcher using the self-registering feature and when an event must be routed it will choose the “best matching target”.

Publish/Subscribe pattern has also been added for outgoing notifications:

- When a notification is sent using a “publish” rule, no destination will be computed, instead an event type identifier will be sent to the message router.
- When the “clients” services connect to the message router, they will announce their subscriptions (by reading their configuration) thus all the message routers will know the list of interested clients and when publishing an event, it will be sent to all the available interested registered services.

Microservice self-monitoring

This covers two new functionalities: 'custom counters' and 'process/service auto-spawn'.

Custom counters can be defined to monitor the processing time for specific events (usually only the request/response/notifications are monitored regardless of the kind of message). Now, a reference time can be defined for the event processing considered as a “normal” processing time for this kind of message for this microservice (ex: order creation on the OMS service). Processing time is computed in real-time and when the service begins to get “overwhelmed”. If the processing time increases over a defined limit, then alerts will be triggered and logged allowing the system to monitor the problem as soon as it occurs and locate the failure directly.

Sometimes, the processing time may increase because the load is getting higher than usual (load peak) so the custom counters can also be linked to the “auto-spawn” feature. If defined inside the configuration, Tango will spawn automatically new instances of the microservice or process whenever the load is going over the predefined limit, allowing the Tango environment to automatically scale in function of the load.

A new microservice will also be added gathering processing times and health status from all the other microservices, allowing centralized monitoring and provide real-time health status information about the environment to a dashboard.

Full Cloud capabilities

In conclusion:

- Adding a new node (or a new service) will auto-register all the services without having to explicitly configure them inside the Tango configuration.
- The load may be balanced to the new node transparently.
- The publish/subscribe pattern allows distributing messages without knowing all the recipients beforehand.

With these new features and its current capacities, Tango can be installed on elastic cloud instances and new nodes can be spawned if the load requires it.

Bibliography:
Microservices – Microservices in action by Morgan Bruce, Paulo Pereira
Microservices – Production-Ready Microservices by Susan J. Fowler
CBSD – An Introduction to Component-Based Software Development by Kung-Kiu Lau, Simone di Cola
Scalability – The Art of Scalability by Martin L. Abbott, Michael T. Fisher

About Lusis Payments

Lusis Payments is a leading global innovator of mission-critical payments software and data science technology. Established in France in 1999, Lusis has offices throughout Europe and North America with customers globally. Lusis is best known for its TANGO solution, an online transaction processing engine for mission-critical 24×7 solutions including payments, retail, loyalty, finance, fraud, utilities, and transport. TANGO delivers performance, availability, and scalability, with a rich set of functionalities, all from a single architecture. TANGO is built on a highly performing micro-service architecture providing agile delivery for business needs. Lusis has also made significant advancements in Data Science, Artificial Intelligence, FX Trading, ISO 20022, Fraud and Blockchain.

lusis Payments

France:

5 Cité Rougemont
75009 Paris
France
(+33) 1 55 33 09 00

Canada:

1 Dundas St West
Suite2500
Toronto, Ontario
Canada, M5G 1Z3

United States Office:

315 Montgomery St.
#900
San Francisco, CA 94104
(+1) 415 829 4577

UK:

1 Northumberland Ave
Trafalgar Square
London, WC2N 5BW
(+44) 207 868 5288

Luxembourg:

321, route d'Arlon
L-8011 Strassen
Luxembourg
(+352) 31 35 02-1

www.lusispayments.com

sales@lusispayments.com

OUR CLIENTS ARE OUR BEST ASSETS

"Our relationship with Lusis has been a tremendous asset to us. We've worked at all levels together...Philippe Preval the President has been a tremendous part of that success...he has a clear grasp of the business and shares our passion for customers."

- Randy Meyer
VP Mission Critical Systems, HP

nuvei Payment Technology Network

"TANGO helps us to provide better value, improved transactional performance and reliability. Our partnership with Lusis also benefits us with added financial efficiencies that allow us to continue enhancing our processing platforms and technologies."

- Philip Fayer
CEO, NUVEI

nets

"...TANGO was the best match for our needs and requirements. Lusis Payments has ensured a successful migration, and the solution is now running excellent."

- Jan Erik Secker

BankservAfrica

"...not only did TANGO deliver the full capability to replace our existing solution but furthermore it delivered on the promises of flexibility, agility, capability and quality."

- Pieter Cilliers
CEO, BankservAfrica