A MAPE-K Approach to Autonomic Microservices

Antonio Bucchiarone, Fondazione Bruno Kessler

Claudio Guidi, italianaSoftware s.r.l.

Ivan Lanese, University of Bologna

Nelly Bencomo, Aston University

Josef Spillner, Zurich University of Applied Sciences

Microservice Conference 2022
Paris, 10-12 May 2022

Outline

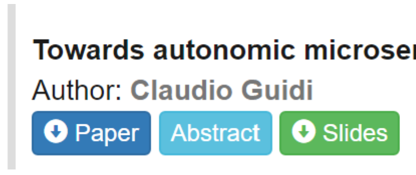
- A brief introduction
  Where this paper comes from
- Present and future scenarios
  Why investigating autonomic microservices?
- A MAPE-K approach
  Our contribution
- Examples and challenges
  How a MAPE-K approach can be useful
- Conclusions

A brief introduction

A step forward

In the previous edition of Microservices Conference 2020...

- Microservices 2020 Conference :



Towards autonomic microservices
Author: **Claudio Guidi**
Paper Abstract Slides

- Started a collaboration with Antonio Buchiarone (FBK), Ivan Lanese (UniBo), Nelly Bencomo (Aston University) and Josef Spillner (Zurich University of Applied Sciences)

- Accepted as a poster at ICSA 2022 New and Emerging Ideas Track

- <u>Accepted for a presentation at Microservices 2022 Conference</u>

- Today we are sharing our ideas with the community in order to stimulate the debate and triggering new research opportunitites

Autonomic Microservices

Autonomic Microservices means applying the concepts of autonomic computing to microservices architectures.

Kephart, Jeffrey & Chess, D.M.. (2003). **The Vision Of Autonomic Computing. Computer**. 36. 41- 50. 10.1109/MC.2003.1160055

Systems manage themselves according to an administrator's goals. New components integrate as effortlessly as a new cell establishes itself in the human body. These ideas are not science fiction, but elements of the grand challenge to create self-managing computing systems.

Concept	Current Computing (2003)	Autonomic computing
Self-configuration	Corporate data centers have multiple vendors and platforms. Installing, configuring, and integrating systems is time consuming and error prone.	Automated configuration of components and systems follows high-level policies. Rest of system adjusts automatically and seamlessly.
Self-optimization	Systems have hundreds of manually set, nonlinear tuning parameters, and their number increases with each release.	Components and systems continually seek opportunities to improve their own performance and efficiency.
Self-healing	Problem determination in large, complex systems can take a team of programmers weeks.	System automatically detects, diagnoses, and repairs localized software and hardware problems
Self-protection	Detection of and recovery from attacks and cascading failures is manual.	System automatically defends against malicious attacks or cascading failures. It uses early warning to anticipate and prevent systemwide failures.

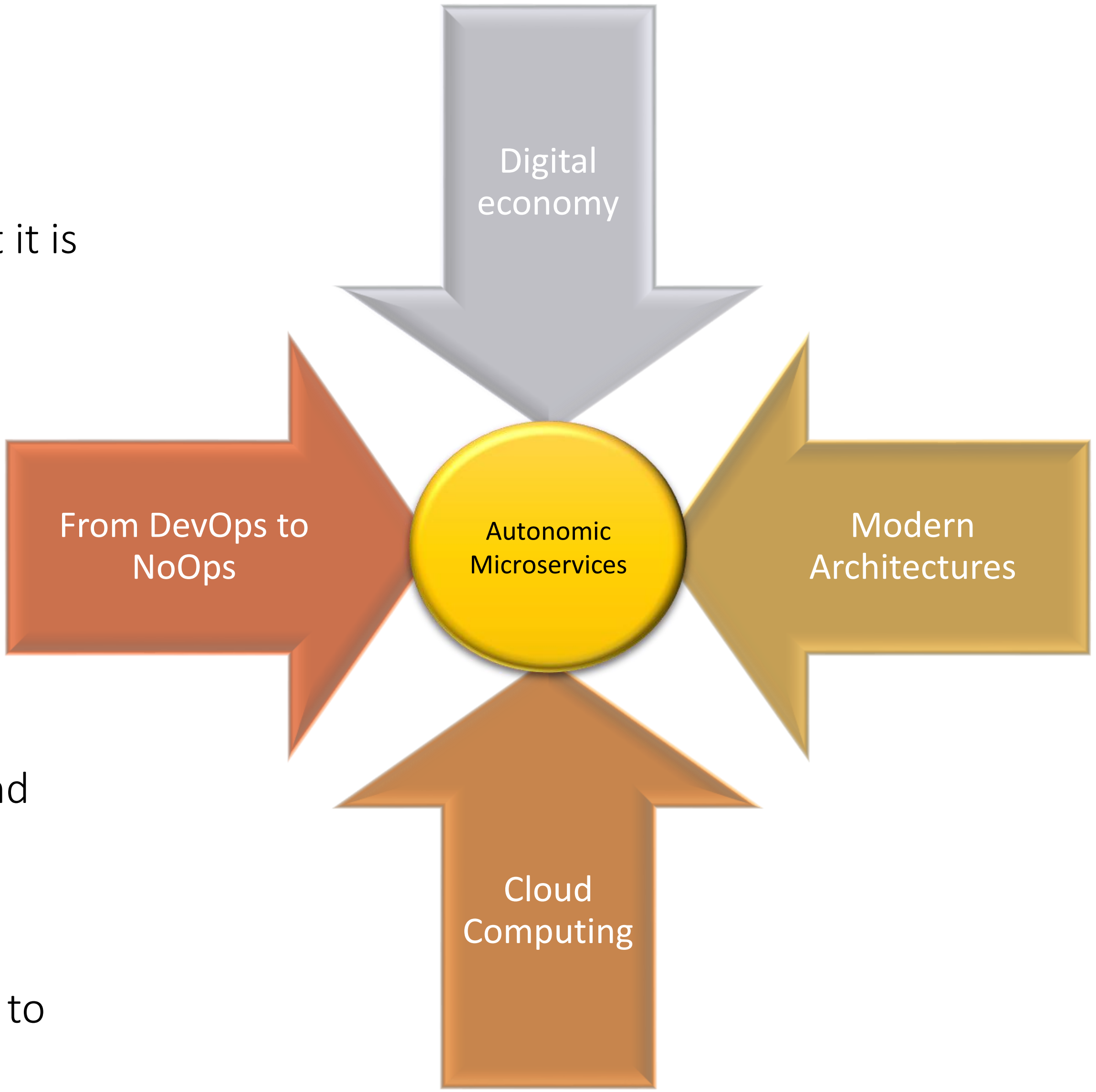
Present and future scenarios

Depicting the scenario

Why investigating autonomic microservices?

Here we identify four main forces that allow us to say that it is important to investigate autonomic microservices:

- Since IT and software infrastructures are assets for the **Digital economy**, it is fundamental to lower their maintenance costs.
- **Cloud Computing** is growing day by day. AI will play an important role in optimizing resources.
- **Modern Architectures** (e.g. microservices) are more and more distributed. The main idea is that, distributed architecture are more resilient, flexible and scalable
- **From DevOps to NoOps** the main idea behind NoOps is to completely automatize the software environment



Digital economy

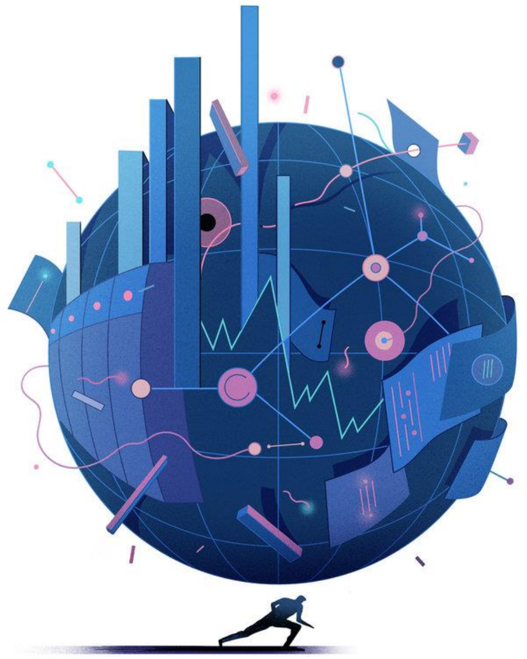
Digital economy is shaping our societies

- There are huge investments of all the countries for addressing the challenges of digital economy. Both public institutions and private players, are pushing towards process digitization

- **Lowering the costs of developing and maintaining software assets is a competitive leverage.**
  - Reducing energy consumption
  - Reducing the need of human presence

FUTURE SCENARIO:
Software infrastructures become so intelligent to self adjust themselves automatically in order to reduce the need of human presence

➡ *We need software infrastructures designed on top of Autonomic Computing foundations*



Cloud computing

Multi cloud and cognitive cloud are the new frontiers

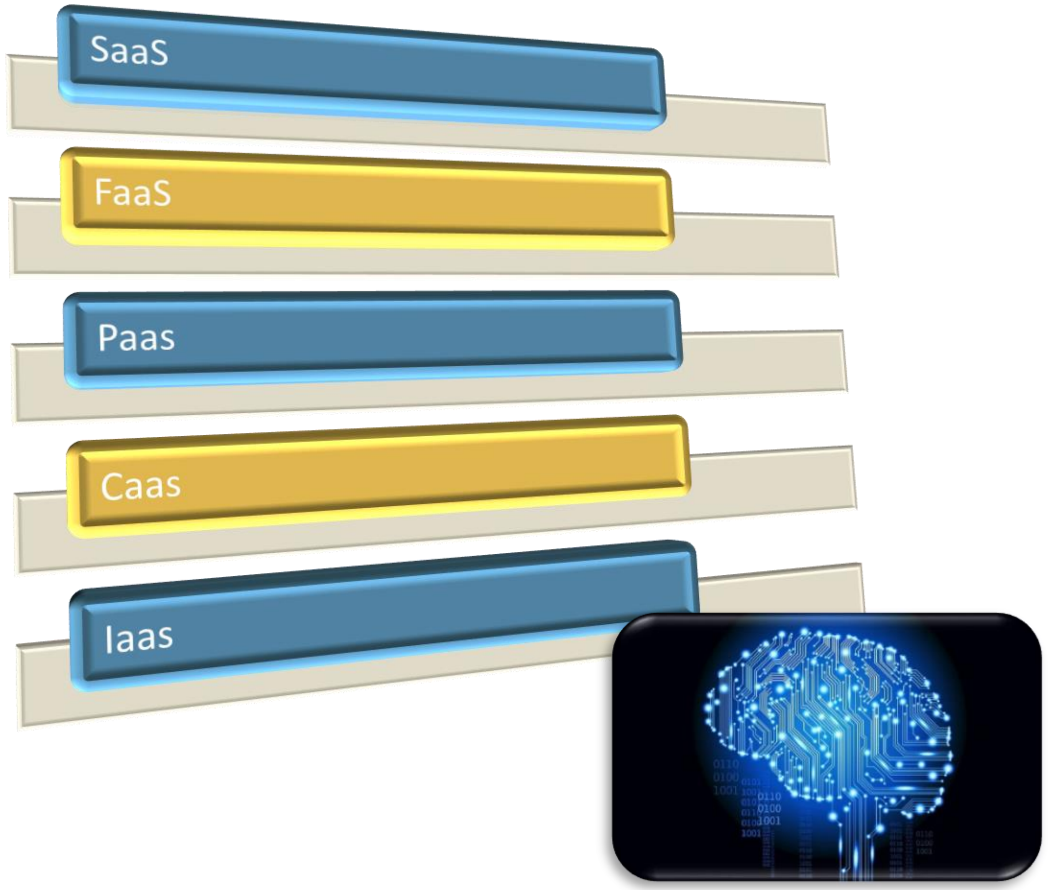
The cloud is the de facto target infrastructure for deploying applications today and in the future, both in the private and public case

- **Cognitive Cloud**: new research directions for equipping cloud computing with **artificial intelligence** used for optimizing resource and energy consumption

FUTURE SCENARIO

Cloud infrastructures will be so intelligent to manage running applications in order to optimize the energy consumption by keeping the same quality

➡ *Making the cloud infrastructure so intelligent could be insufficient if the applications are not intelligent too.*



Modern architectures

Distributed architectures are taking place everywhere

Microservices and **serverless** architectures are taking place for designing and developing new applications.

The main objectives are:

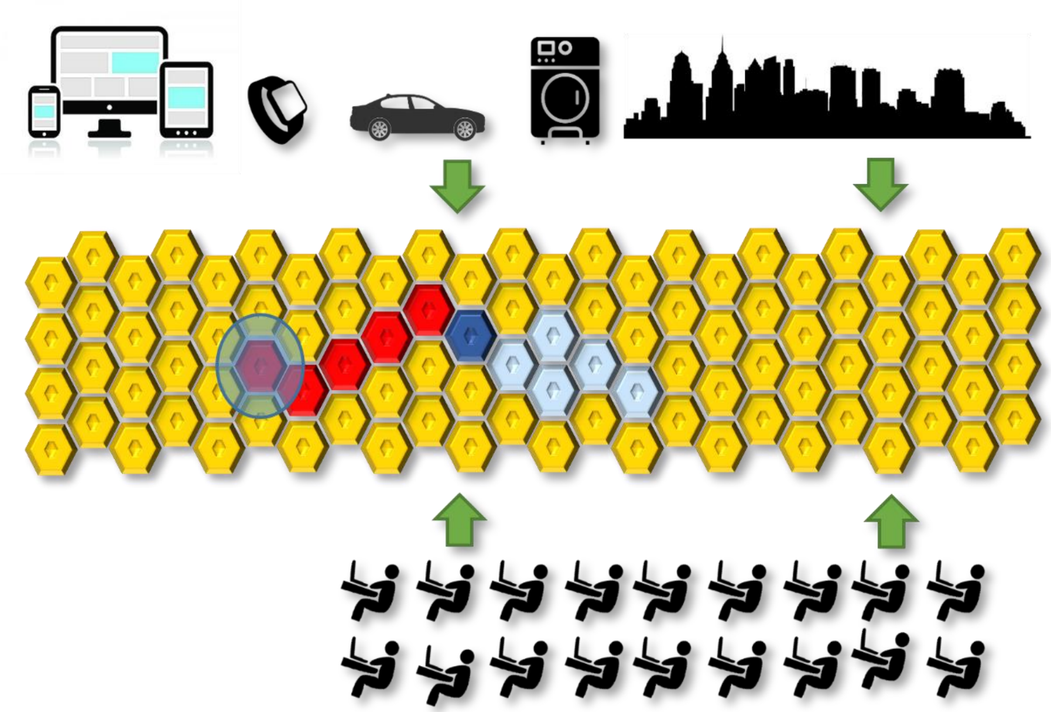
- creating **scalable** and **flexible** applications;
- creating reusable components;
- technology agnosticism.

See also the vision of «The composable Enterprise» from Gartner

FUTURE SCENARIO

All the applications will be developed as a composition of distributed components deployed in a multi cloud infrastructure

The overall complexity will increase dramatically. Autonomic distributed applications will help in the management of such a complexity.

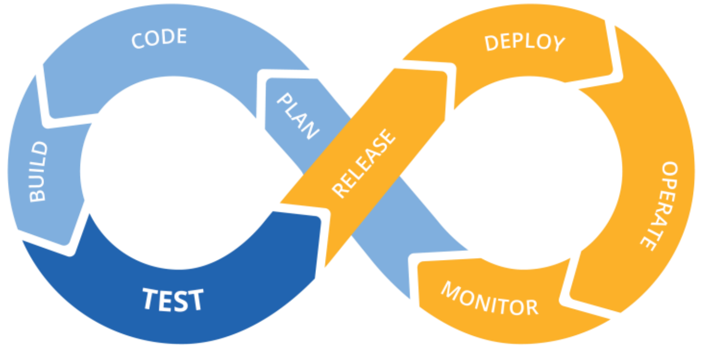


From DevOps to NoOps

Full automatization of software production chain

DevOps processes automatize several aspects of the software production chain. The next step is NoOps.

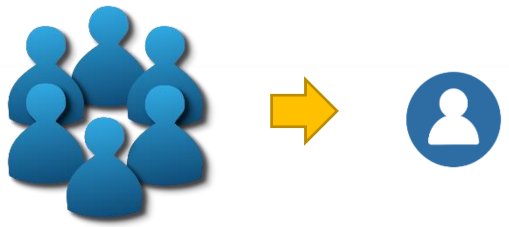
With NoOps the software environment is completely automated that there's no need for an operations team to manage it.



FUTURE SCENARIO

The NoOps dream comes true, the software environment does not need to human intervention anymore.

Autonomic computing is a fundamental piece of NoOps, because it allows to program the maintenance tasks as the autonomic behavior of the applications.



Waiting for the future…

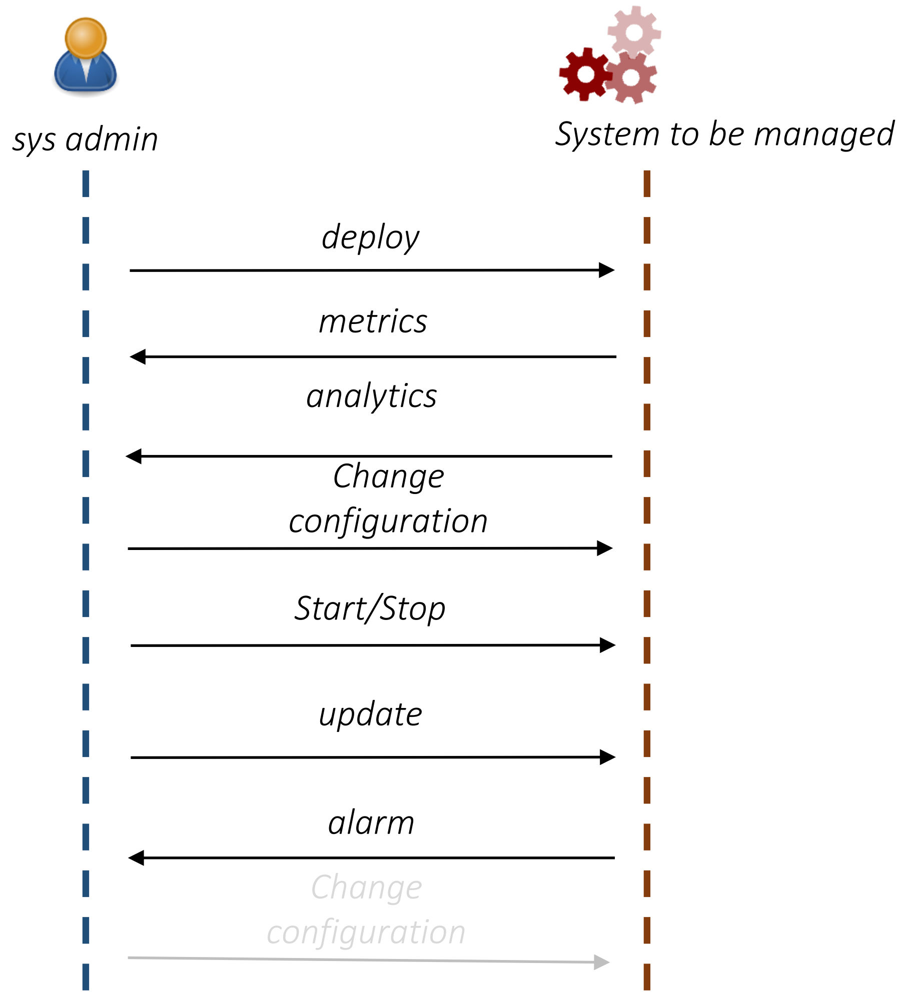
Research on autonomic microservices can bring other benefits

- Autonomic microservices can help in defining and standardizing non-functional requirements for microservices, thus permitting to avoid cloud vendor lock-in.
  As an example, some of the following features are often coupled with the underlying infrastructure:
  - *Observability*
  - *Scalability*
  - *Configurability*
  - *…*
- Autonomic microservices can help in standardizing the containerization layer of cloud infrastructure, thus facilitating the adoption of standard API for automatically negotiating resources with cloud providers

A MAPE-K approach for autonomic microservices

Autonomic computing and microservices

Visualizing the idea of autonomic computing in a microservices environment

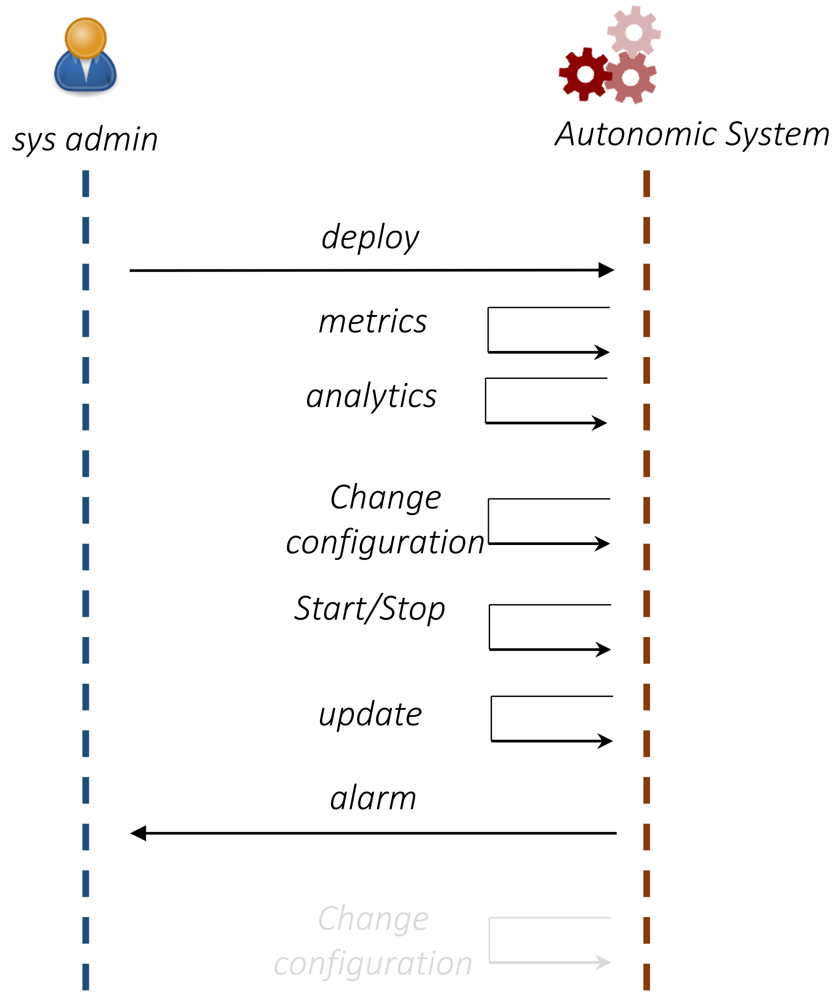


The standard way for managing a system

The management of a running system can be seen as a continuous set of interactions between the sys admin and the target infrastructure in order to keep the applications running with a high level of quality targeting the business requirements.

Autonomic computing and microservices (2)

A lot of interactions between admin and the execution environment are automatically managed by the system itself



Ideally, an autonomic system is able to self-configure, self-heal, self-protect and self-organize itself depending on its status.

The main target is reducing human interactions in order to reducing time cost and increasing efficiency.

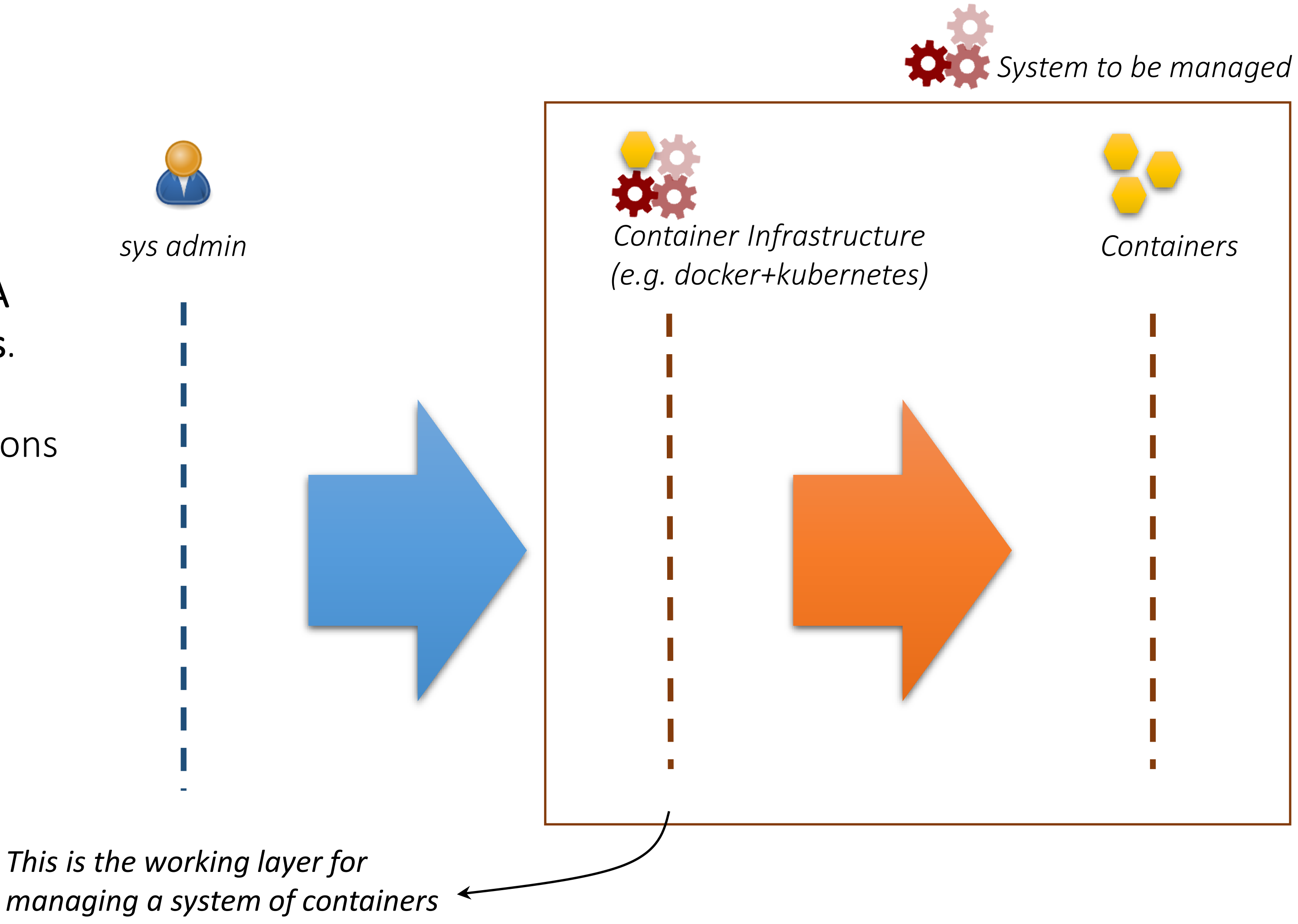
Autonomic computing and microservices (3)

The execution environment can be seen as the composition of a specialized container management infrastructure and the actual containers.

Containerization enables component abstraction to containers and changes the rules on how a system is managed today. **A system is just a set of interacting containers**.

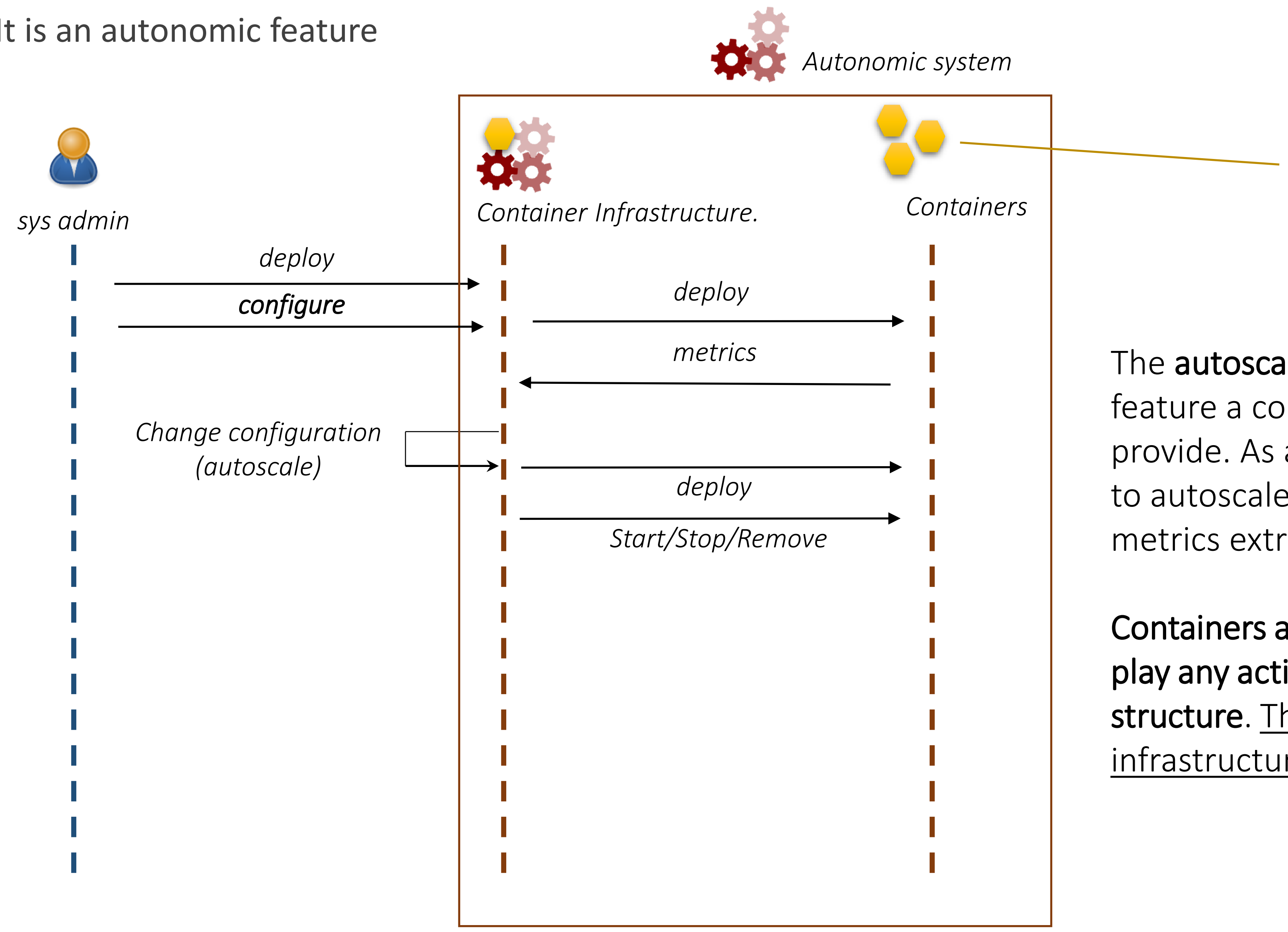
Sys admins can automatize a lot of operations by configuring the orchestration platform.

.


System to be managed
sys admin
Container Infrastructure (e.g. docker+kubernetes)
Containers
This is the working layer for managing a system of containers

Autoscaling

It is an autonomic feature



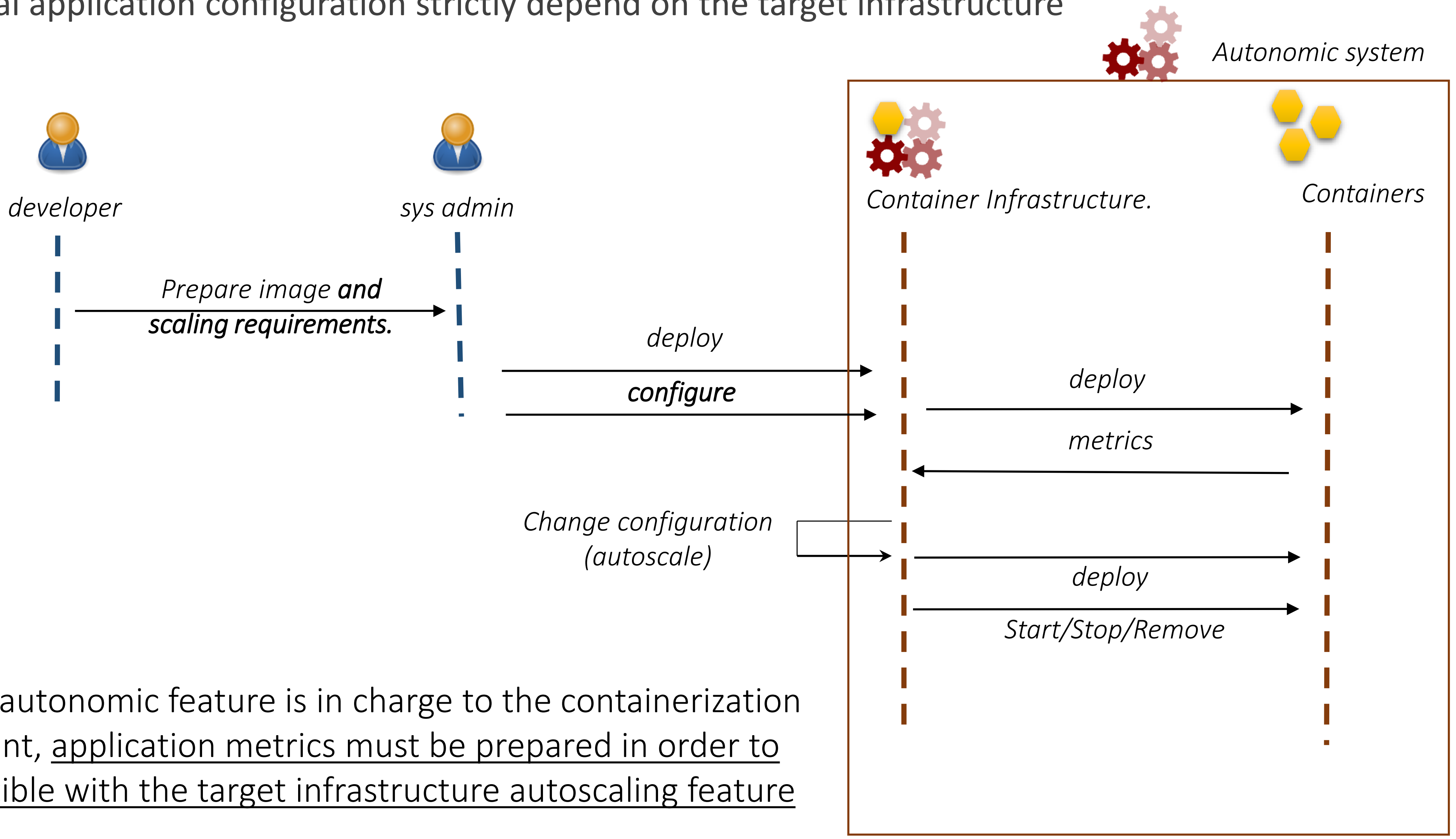
They are not autonomic

The **autoscaling** is maybe the first autonomic feature a containerization system can provide. As an example, Kubernetes is able to autoscale a system depending on some metrics extracted from single components.

Containers are just components that cannot play any actions for changing their own structure. <u>They are manipulated by the infrastructure</u>. They are not autonomic.

The risk of coupling

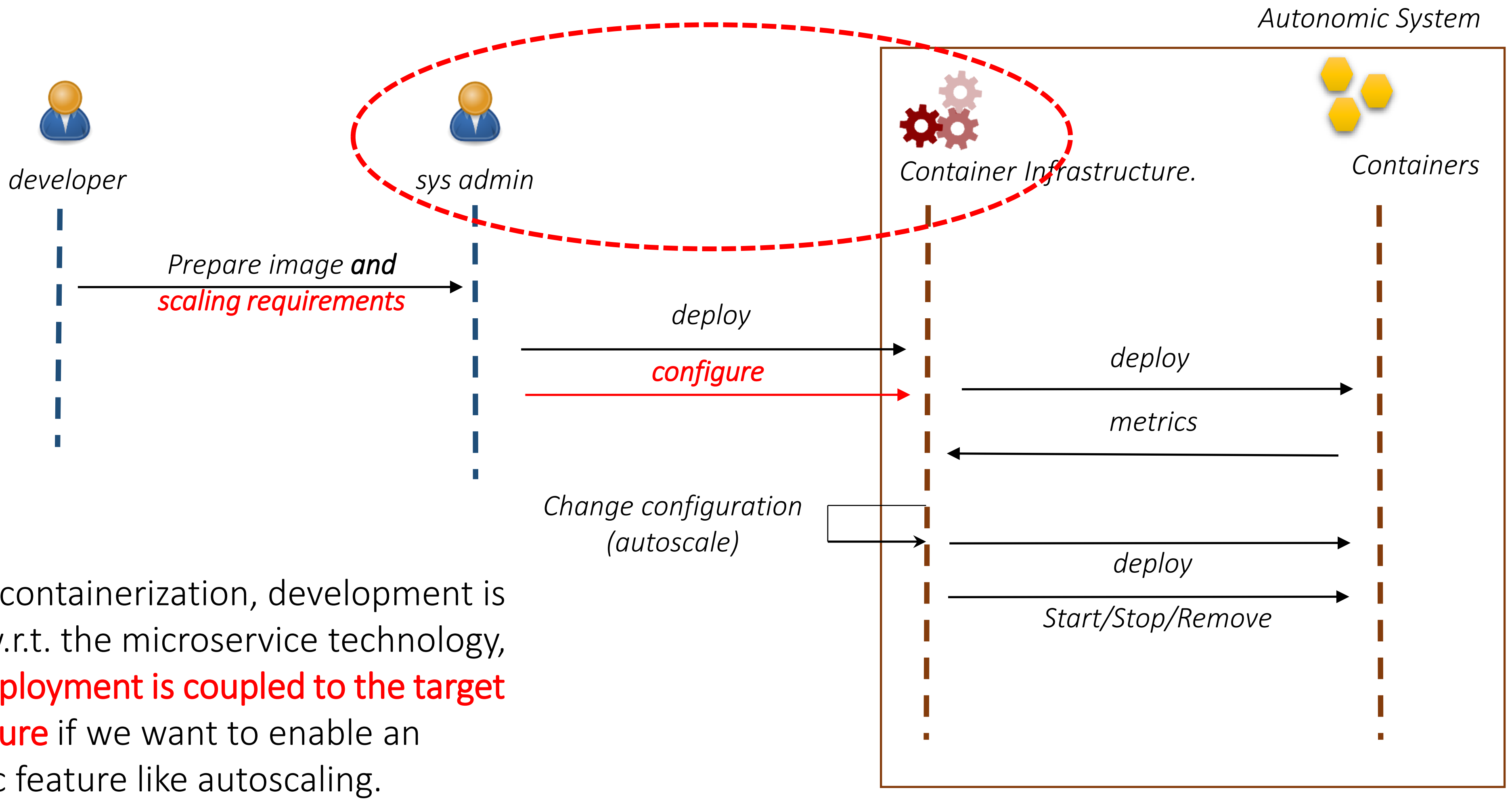
The final application configuration strictly depend on the target infrastructure


Autonomic system
developer
sys admin
Container Infrastructure.
Containers
Prepare image **and scaling requirements.**
deploy
configure
deploy
metrics
Change configuration (autoscale)
deploy
Start/Stop/Remove

When the autonomic feature is in charge to the containerization environment, application metrics must be prepared in order to be compatible with the target infrastructure autoscaling feature

The risk of coupling (2)

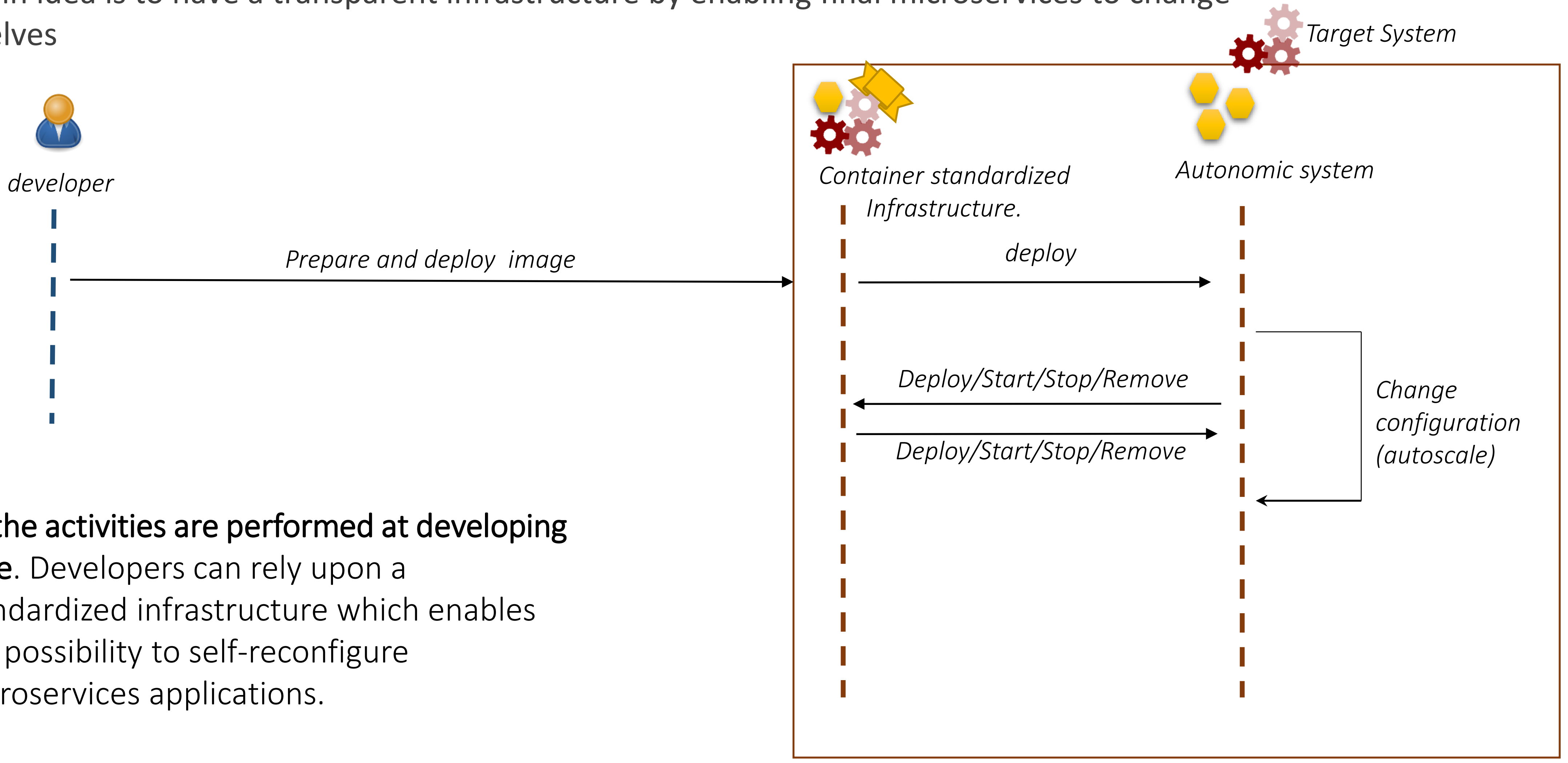
The final application configuration strictly depend on the target infrastructure



Thanks to containerization, development is agnostic w.r.t. the microservice technology, but the deployment is coupled to the target infrastructure if we want to enable an autonomic feature like autoscaling.

Autonomic microservices

The main idea is to have a transparent infrastructure by enabling final microservices to change themselves

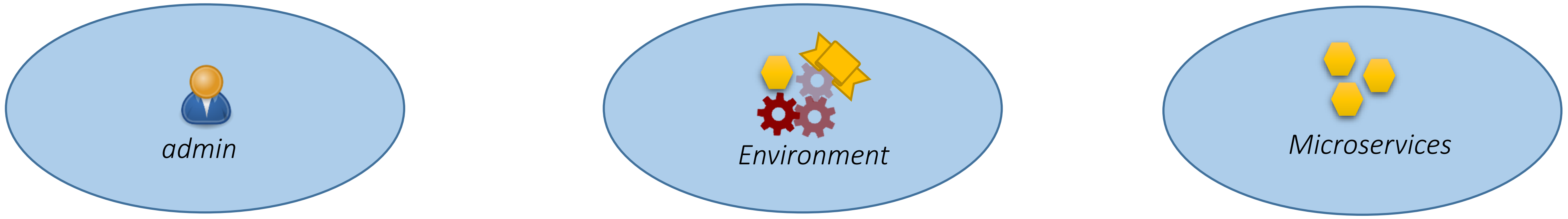


All the activities are performed at developing time. Developers can rely upon a standardized infrastructure which enables the possibility to self-reconfigure microservices applications.

The main concepts we extract

These concepts are at the basis of our proposal

1. There are different **responsibilites**: human, environment and microservices



2. There are different actions to be taken:
   1. Monitoring
   2. Analyzing metrics
   3. Planning the next action
   4. Executing the action

A MAPE-K approach

A proposal for catching the complexity

In our proposal we exploit the **feedback loops approach** used in the area of self-adaptive systems, as a means for describing the different responsibilities.

The MAPE-K feedback control loop performs Monitor-Analyze-Plan-Execute phases over a shared Knowledge.

- The **Monitoring** (M) phase acquires data from the system and its environment.
- **Analysis** (A)of the monitored data involves activities such as filtering or transforming data, e.g. to reduce noise or to put them in a form suitable for elaboration.
- Then, **Planning** (P) of future actions is done, keeping into account the result of the analysis as well as the knowledge of the system held in the model.
- Finally, the **Execution** (E) of the planned actions should be performed. This consists of changing the values on the actuator(s) in the system according to the developed plan.
- A MAPE-K loop stores the **Knowledge** (K) required for decision-making in what is called the Knowledge Base (KB).

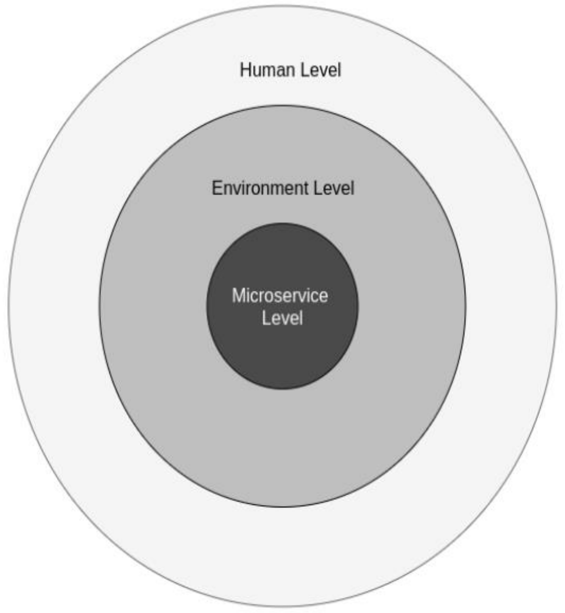
Introducing the graphical notation

Depicting the responsibilities

When integrating a MAPE-K loop in a microservice architecture there are various options related to **which part of the system performs the different phases involved in the loop**.

Possibilities include having phases performed by the infrastructure (e.g., Containers, the Cloud, etc.), by each microservice in addition to its functionalities, by ad-hoc microservices, or by groups of microservices.

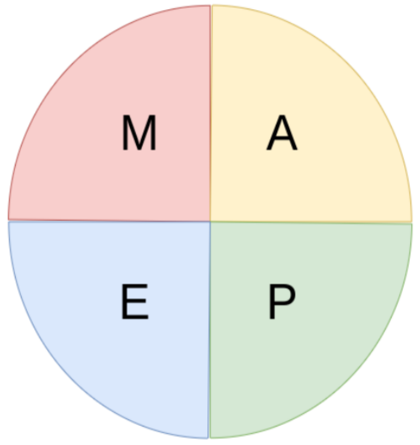
In order to better discuss the various possibilities, we introduce a graphical notation to provide at-a-glance an informative view of the main design decisions, in particular hinting who is in charge of the different phases of the MAPE-K loop. It does not aim at conveying all the details of the chosen approach but more at highlighting the main design choices.



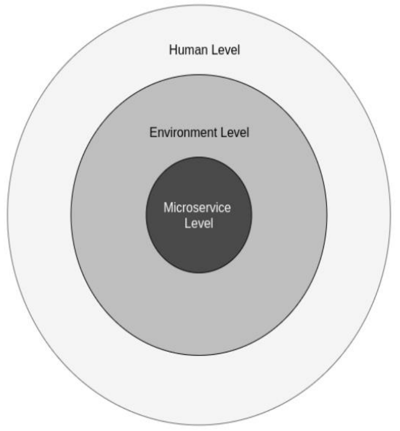
(a) Levels of responsibilities in autonomic microservices.

Introducing the graphical notation

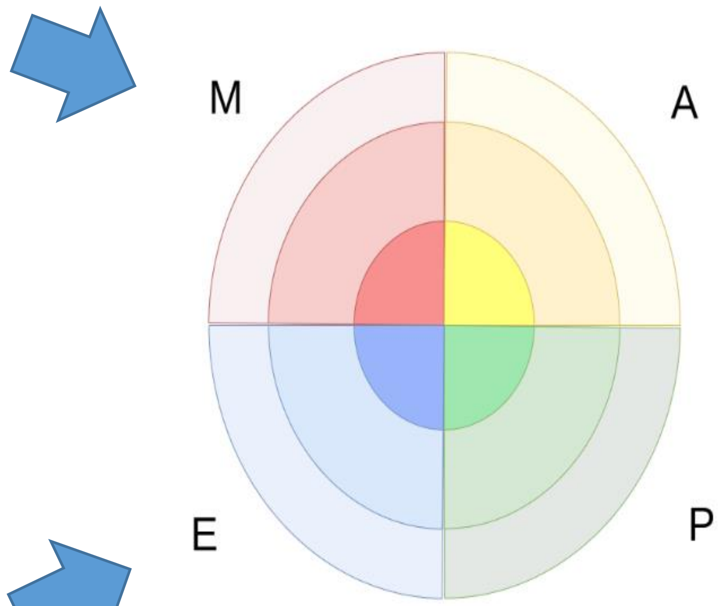
Detailing the responsibilities in a MAPE loop



(b) MAPE phases as sectors.



(a) Levels of responsibilities in autonomic microservices.



(c) Autonomic microservice landscape.

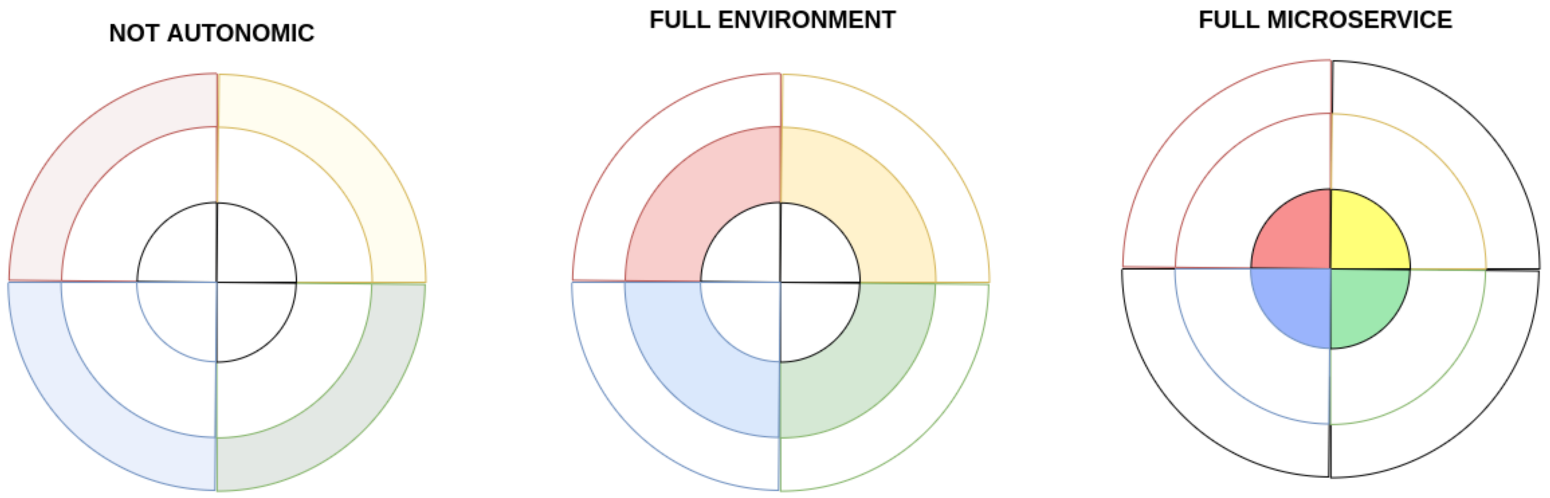
Essentially, each area of the figure represents whether the actor corresponding to the circle takes some responsibility for the activity corresponding to the MAPE sector:

- if the sector is white then it takes no responsibility;
- if it is colored then the actor has some responsibility.
- for each MAPE sector at least one segment (i.e., portion of circle in the sector) needs to be colored since at least one actor must take the responsibility for the activity.
- more than one segment can be colored in a sector, since multiple actors can cooperate in taking responsibility for the activity.

Note that we do not explicitly represent the knowledge base: in many cases it is under the responsibility of the entity that takes care of the planning.

Special cases

Capturing the autonomic degree of a system at a glance

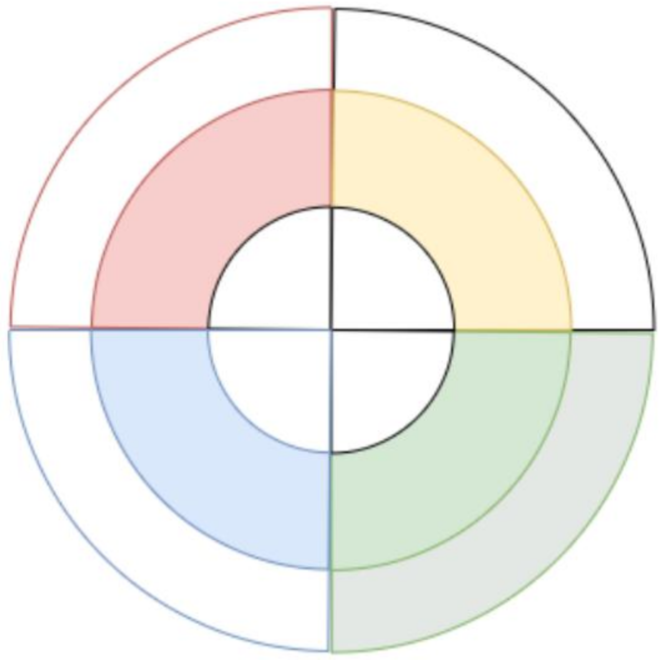


Let’s note that systems more colored towards the center are more autonomic than systems more colored at the border.

We argue that such a diagram can help architects, developers and sysadmins to grasp at the glance the degree of autonomicity of a running microservice system, as well as the main responsibilities involved in the chosen approach. Further, the same notation can also be used at the level of system design to highlight the main requirements

Self scaling with Kubernetes

Auto scaling is a feature of Kubernetes

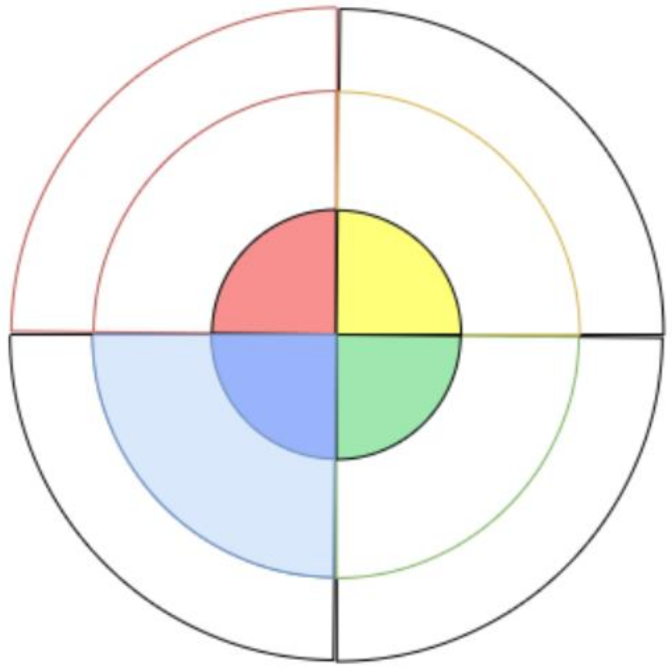


Referring to a Kubernetes-based scenario, auto-scaling would be represented in our scale diagram as an environment-based scenario, where:

- **Monitoring** is performed by Kubernetes only, which continuously collects metrics from the deployed pods.
- **Analysis** is also performed by Kubernetes, which extracts statistics from the metrics.
- **Planning** is performed by Kubernetes as well, depending on the initial customization provided by the sysadmin. Here we highlight also the sector of human responsibility, as planning requires a previous configuration performed by a human actor. *Only when the applied planning is standard and always applied without customization, planning can be put in charge exclusively to the environment.*
- **Execution** is provided by Kubernetes, which autonomously scales the components and takes care of load balancing.

A proof of concept

At Microservices 2020, a jolie based proof of concept was presented


(f) Proof of concept.

Here a functionality is originally provided by a single jolie microservice deployed with a docker environment. Then, a load increment and a corresponding slow down in the response time are simulated on its listening endpoints. The microservice, **that is autonomously able to detect a deterioration of the response time of its operations**, then invokes the environment to scale up one of its core microservices into the current infrastructure. After some delay, the load is simulated to decrease thus improving the response time, resulting in the microservice asking for scaling down the extra instances.

• the microservice implements a simple monitor of its operations. In particular, it collects the response time average. Thus, phase M is under its responsibility;
• the microservice detects a decrease in response time due to high load. Thus, phase A is under its responsibility;
• the microservice decides when to ask for a new instance for scaling the load. Thus, phase P is under its responsibility;
• the microservice negotiates with the environment the release of a new instance. Thus, **phase E is under shared responsibility of the microservice and the environment**.
• the microservice holds all the knowledge about the component to be deployed. The infrastructure is not aware of the new component to be deployed neither it is aware of its container image, which is built at runtime.

Challenges and opportunities

We identify four main challenges that must be overcome in order to implement our approach

1. **Standardised discoverable APIs.** Autonomic Microservices require in many cases interaction between the services and the infrastructure. This is the case at least every time a phase is a shared responsibility between the two. Such an interaction should take place via well-defined APIs. In order to allow deployment on multiple infrastructures and to reduce vendor-lockin **such APIs should be standardized**, so that the same interface would be available on different platforms. Ideally, custom implementations of the required functionalities could be provided whenever the infrastructure does not cover them. We remark that a decision to use the MAPE-K control loop to approach autonomicity could give guidelines on how to define such a standardized API, which should reasonably involve functionalities for each of the 4 phases and for sharing the knowledge base K.

2. **Intent-based specification of autonomicity**. Rather than specifying imperatively all technical instructions on which change to perform under which condition, high-level intents or rules (i.e. described instead of prescribed) that keep into account both technical aspects (e.g., throughput) and economic concerns (e.g., price of resources) are to be preferred. This allows one to state objectives such as ”any change is allowed as long as the cost of service operation does not exceed a maximum threshold”. As usual, moving from an imperative to a declarative approach makes the description of the autonomic behaviour more flexible and easy to understand, yet the translation of such a specification to code executable on top of the available API becomes more complex.

Challenges and opportunities

We identify four challenges that must be overcome in order to implement our approach

3. **Multi-cloud support**. If autonomous services are meant to automate human logic on what to run where, this includes necessarily decisions on when to activate additional providers or migrate between providers. For this to be technically feasible, the following information needs to be available: shared knowledge on what providers exist and what as-a-service models they support with which non-functional properties

4. **Ecosystem of MAPE-K logic.** Many problems that are addressed by autonomy are shared across application domains. Engineers will benefit from accessing libraries and repositories containing custom logic for the analysis of monitoring data and the planning of next steps. This logic could itself be offered in the form of microservices, such as stateless functions delivering analytical results and planning decisions, for technological coherence and increased chance of adoption.

Conclusions

Conclusions

Triggerring a discussion within the microservices community

Autonomic microservices are very challenging. Our contribution is to provide a general framework for addressing their design and development by highlighting some important aspects to be taken into account:

- A clear definition of the involved actors and their responsibilities
- A reference model to be used (MAPE-K loop)

Thanks