Università di Catania

Uni ct MATEMATICA E INFORMATICA

Introduzione a DevOps

Alessandro Midolo, PhD

Dipartimento di Matematica e Informatica

Università di Catania

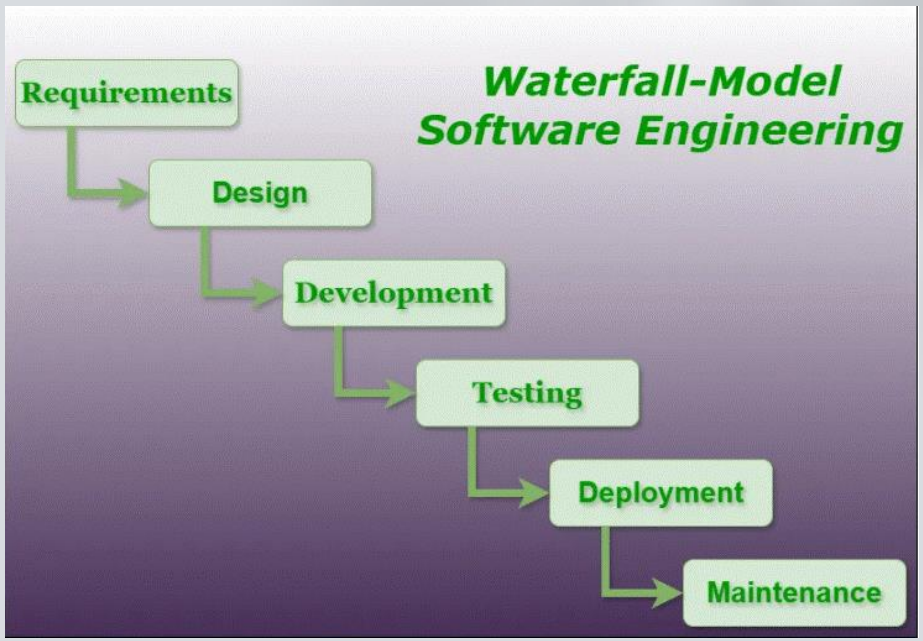
alessandro.midolo@unict.it

https://www.dmi.unict.it/amidolo/

A.A. 2024/2025

Il modello a Cascata (Waterfall)

- Il processo di sviluppo seguiva **fasi** lineari
  - analisi dei requisiti
  - Progettazione
  - Implementazione
  - Testing
  - Distribuzione
  - manutenzione
- Ogni fase doveva essere completata prima di passare alla successiva
- Lo sviluppo del software era tipicamente **separato** dalle operazioni IT.
- Lo sviluppo era un processo **rigido** e spesso **inefficiente**, con **errori** o **incomprensioni** sulle configurazioni dell'ambiente



Il modello Agile

- Iterazioni brevi (Sprint)
- Consegna continua di valore
- Collaborazione costante
- Adattabilità
- Feedback continuo
- Self-organizing teams

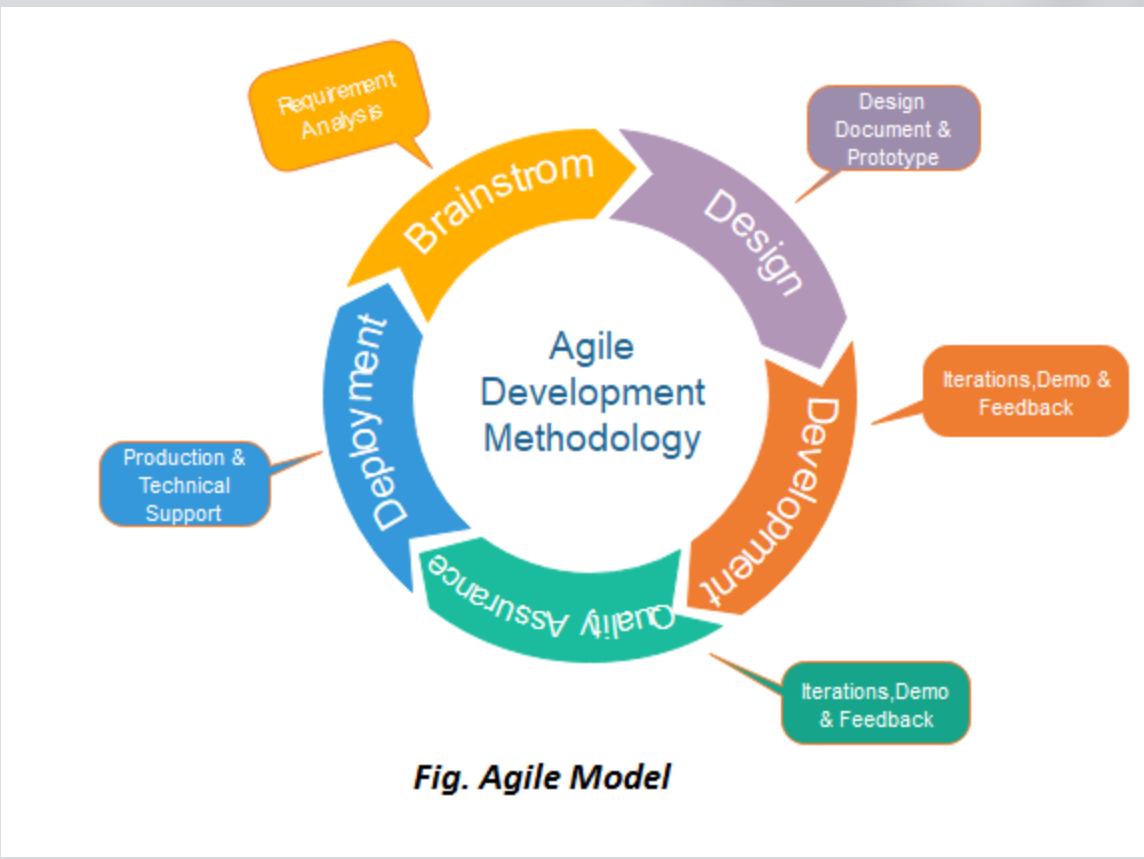


Fig. Agile Model

Agile vs DevOps

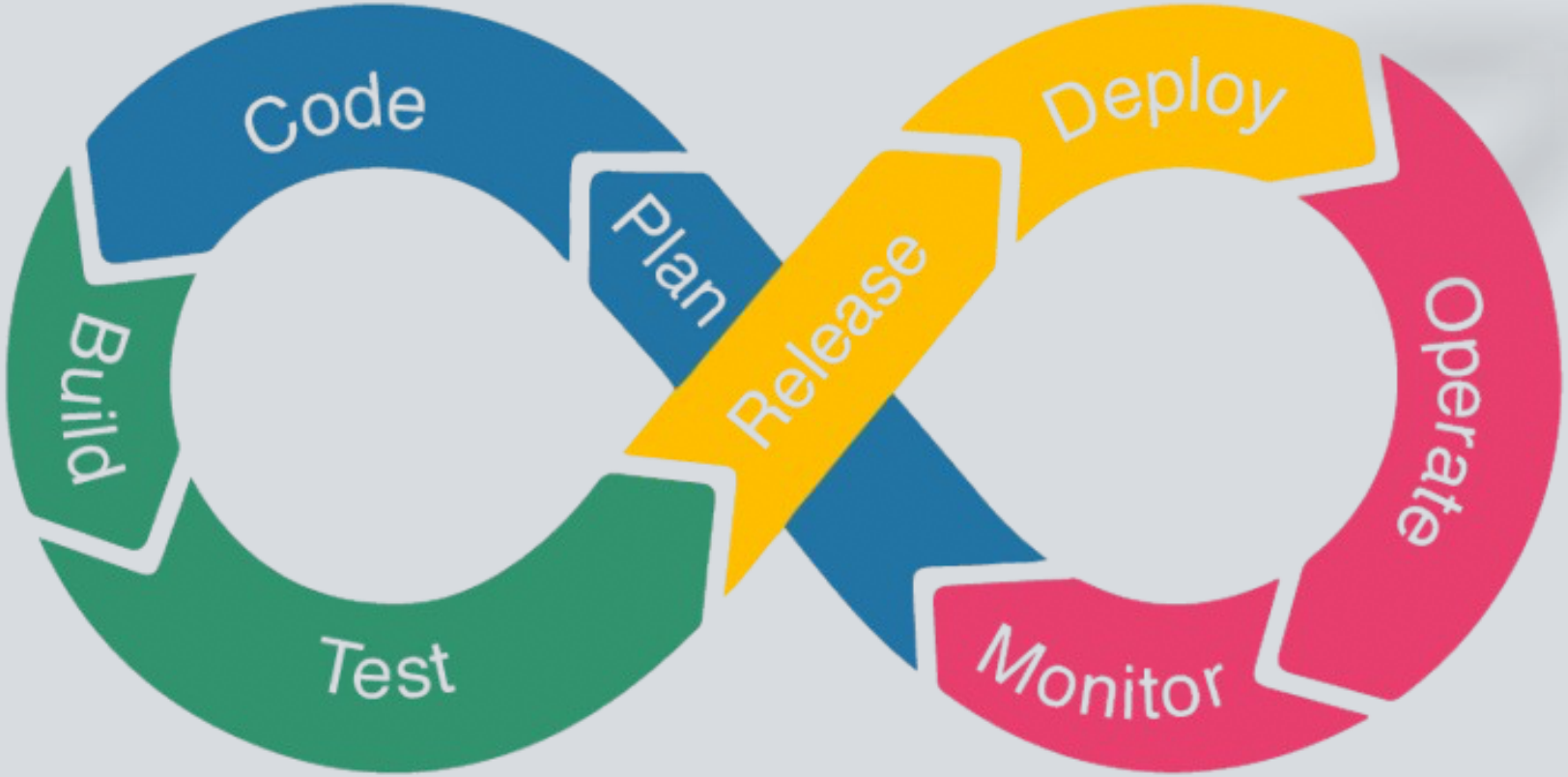
- **Agile** si concentra principalmente sul processo di sviluppo, mentre **DevOps** estende i principi Agile all'intero ciclo di vita del software, inclusa la fase operativa
- DevOps adotta i cicli rapidi di sviluppo di Agile e li unisce con **l'automazione** e la **collaborazione** tra i team di sviluppo e operazioni, per assicurare che il software venga rilasciato rapidamente e in modo affidabile


Waterfall Model
DevOps Model

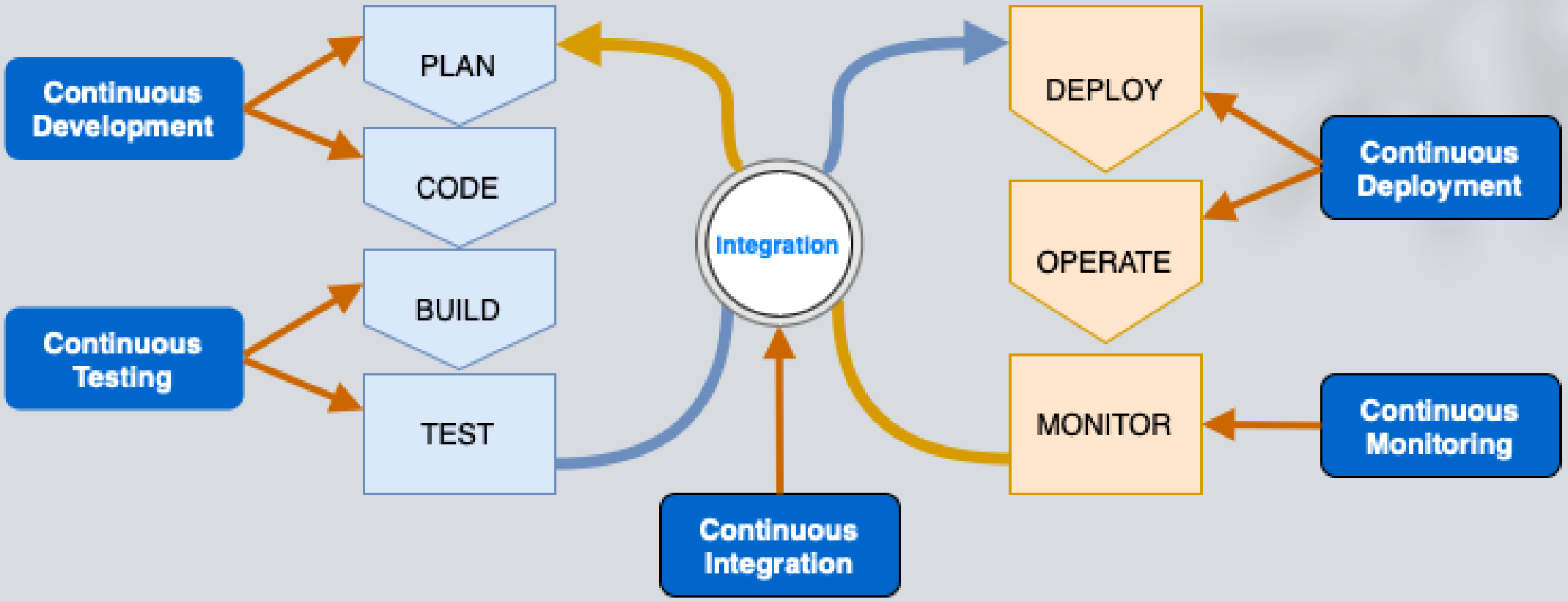
Development & Operations

- Il "DevOps" è una combinazione di pratiche, strumenti e filosofie culturali che ha l'obiettivo di migliorare la **collaborazione** tra i team di sviluppo software (Development, o "**Dev**") e quelli di gestione delle operazioni IT (Operations, o "**Ops**")
- DevOps cerca di automatizzare e integrare i processi tra questi team, permettendo loro di **costruire**, **testare** e **rilasciare** software in modo più rapido, affidabile e continuo.

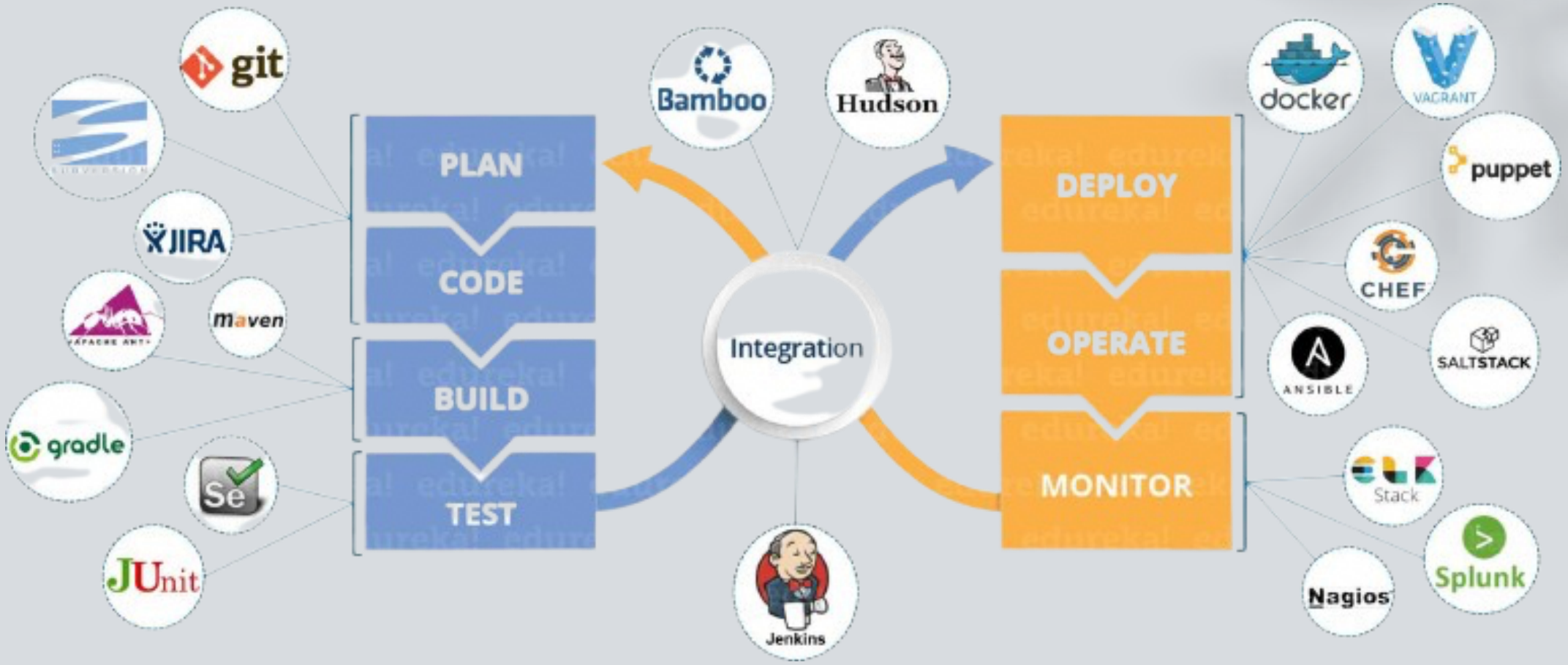
DevOps Lifecycle



DevOps Phases



DevOps Tools



Source: https://www.edureka.co/blog/devops-tools

Punti Chiave

- Source Code Management (SCM) & pipeline
- Continuous Integration
- Continuous Delivery
- Microservices
- Infrastructure as a code (automation)
- Monitoring e Logging
- Communication and Collaboration

SCM Pipelines

- Le componenti principale di una pipeline sono:
  - **Stages** (fase) rappresentano un insieme logico di attività. Le fasi vengono eseguite in un ordine specifico, una dopo l'altra (e.g. prima si esegue la fase di build, poi quella di test, e infine quella di deploy)
  - **Jobs** sono singoli task che eseguono una specifica operazione (compilazione del codice, esecuzione dei test, deployment). I jobs all'interno della stessa fase vengono eseguiti in parallelo
  - **Runners** sono i processi che eseguono i jobs delle pipelines. I runner possono essere eseguiti su macchine virtuali, container, o fisici, e supportano diversi ambienti
  - **Triggers** le pipelines possono essere attivate in vari modi: automaticamente (ad esempio, dopo un commit), tramite merge request, o manualmente.
  - **Artifacts** sono file generati dai jobs che possono essere conservati e utilizzati da altre fasi della pipeline. Ad esempio, il risultato della fase di build può essere usato nelle fasi di test o deploy.

File di Configurazione

- Il file **.gitlab-ci.yml** è il cuore della configurazione CI/CD di GitLab
- In esso vengono definite le fasi della pipeline, i jobs all'interno di ciascuna fase, le condizioni di esecuzione, gli script che vengono eseguiti, le variabili di ambiente e tanto altro

Source
https://docs.gitlab.com/ee/ci/pipelines/
https://docs.gitlab.com/ee/ci/examples/

Continuous Integration

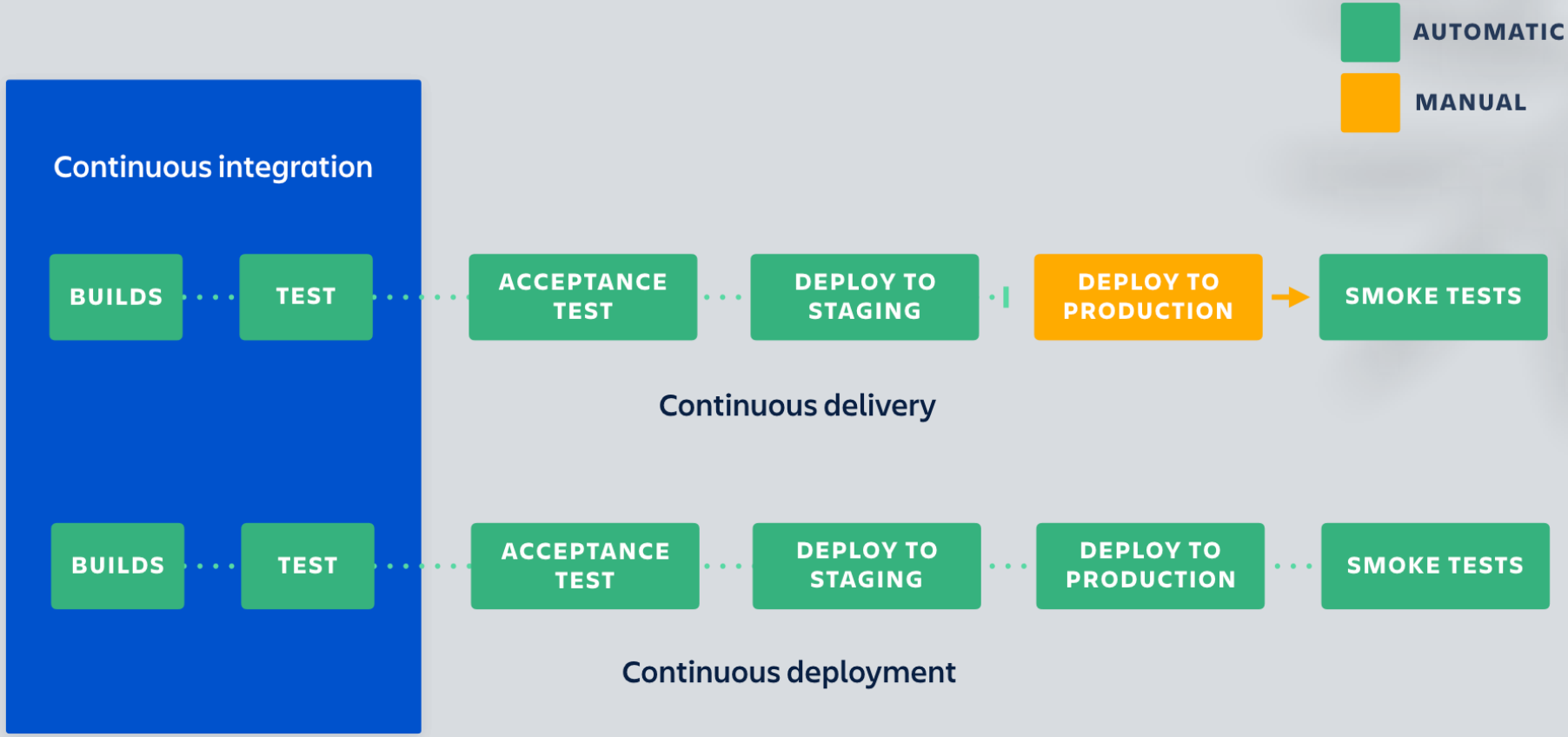
- Gli sviluppatori che praticano il **continuous integration** uniscono le loro modifiche al branch principale il più spesso possibile
- Il commit scatena l'esecuzione **automatica** della compilazione e dei test (unità e integrazione)
  - Rilevare rapidamente eventuali **difetti** nel branch principale
  - Migliorare la **qualità** del codice
  - Diminuire i tempi di **validazione** e **rilascio** dei nuovi aggiornamenti

Continuous Delivery

- E’ un'estensione del continuous integration, poiché **distribuisce** automaticamente tutte le modifiche al codice in un ambiente di test e/o produzione dopo la fase di compilazione
- Ciò significa che oltre ai test automatizzati, si ha un processo di **rilascio automatizzato** e puoi distribuire la tua applicazione in qualsiasi momento cliccando su un pulsante
- **Rilasci** giornalieri, settimanali, etc.
- Conviene distribuire in produzione il prima possibile per assicurare di rilasciare **piccoli lotti** che siano facili da risolvere in caso di problemi

Continuous Deployment

- La **Continuous Deployment** va un passo oltre la Continuous Delivery
- Ogni modifica che supera tutte le fasi della **pipeline** di produzione viene rilasciata ai clienti
- Non c'è alcun **intervento umano** e solo un test non riuscito impedirà che una nuova modifica venga distribuita in produzione
- E' un modo eccellente per **accelerare** il ciclo di feedback con i clienti e togliere pressione al team poiché non esiste più un «giorno di rilascio»
- Gli sviluppatori possono concentrarsi sulla **creazione di software** e vedono il loro lavoro andare in diretta pochi minuti dopo aver finito di lavorarci



Source: https://www.atlassian.com/continuous-delivery/principles/continuous-integration-vs-delivery-vs-deployment

Moving Fast with Software Verification

Cristiano Calcagno, Dino Distefano, Jeremy Dubreil, Dominik Gabi, Pieter Hooimeijer, Martino Luca, Peter O'Hearn, Irene Papakonstantinou, Jim Purbrick, and Dulma Rodriguez

Facebook Inc.

Abstract. For organisations like Facebook, high quality software is important. However, the pace of change and increasing complexity of modern code makes it difficult to produce error-free software. Available tools are often lacking in helping programmers develop more reliable and secure applications.

Formal verification is a technique able to detect software errors statically, before a product is actually shipped. Although this aspect makes this technology very appealing in principle, in practice there have been many difficulties that have hindered the application of software verification in industrial environments. In particular, in an organisation like Facebook where the release cycle is fast compared to more traditional industries, the deployment of formal techniques is highly challenging.

This paper describes our experience in integrating a verification tool based on static analysis into the software development cycle at Facebook.

1 Introduction

This is a story of transporting ideas from recent theoretical research in reasoning about programs into the fast-moving engineering culture of Facebook. The context is that most of the authors landed at Facebook in September of 2013, when we brought the INFER static analyser with us from the verification startup Monoidics [4, 6]. INFER itself is based on recent academic research in program analysis [5], which applied a relatively recent development in logics of programs, separation logic [10]. As of this writing INFER is deployed and running continuously to verify select properties of every code modification in Facebook's mobile apps; these include the main Facebook apps for Android and iOS, Facebook Messenger, Instagram, and other apps which are used by over a billion people in total.

In the process of trying to deploy the static analyser the most important issue we faced was integration with Facebook's software development process. The software process at Facebook, and an increasing number of Internet companies, is based on fast iteration, where features are proposed and implemented and changed based on feedback from users, rather than wholly designed at the outset. The perpetual, fast, iterative development employed at Facebook might seem to be the worst possible case for formal verification technology, proponents of which sometimes even used to argue that programs should be developed only

Calcagno, Cristiano, et al. "Moving fast with software verification." *NASA Formal Methods Symposium*. Cham: Springer International Publishing, 2015.

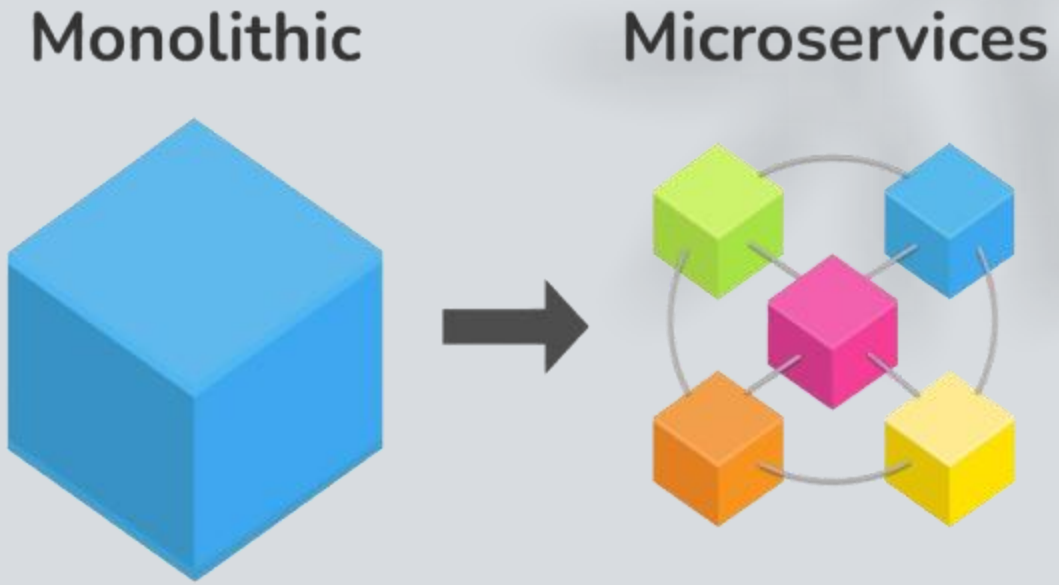
Monolithic

- Prevede che l'intera applicazione sia **un'unità singola** e **indivisibile**, dove tutte le funzionalità (frontend, backend, logica di business, database, ecc.) vengono eseguite all'interno di un unico processo e distribuite come un blocco unico
- Difficoltà di gestione con il crescere dell'applicazione
- Scalabilità limitata
- Aggiornamenti complessi

And here it comes…. MICROSERVICE

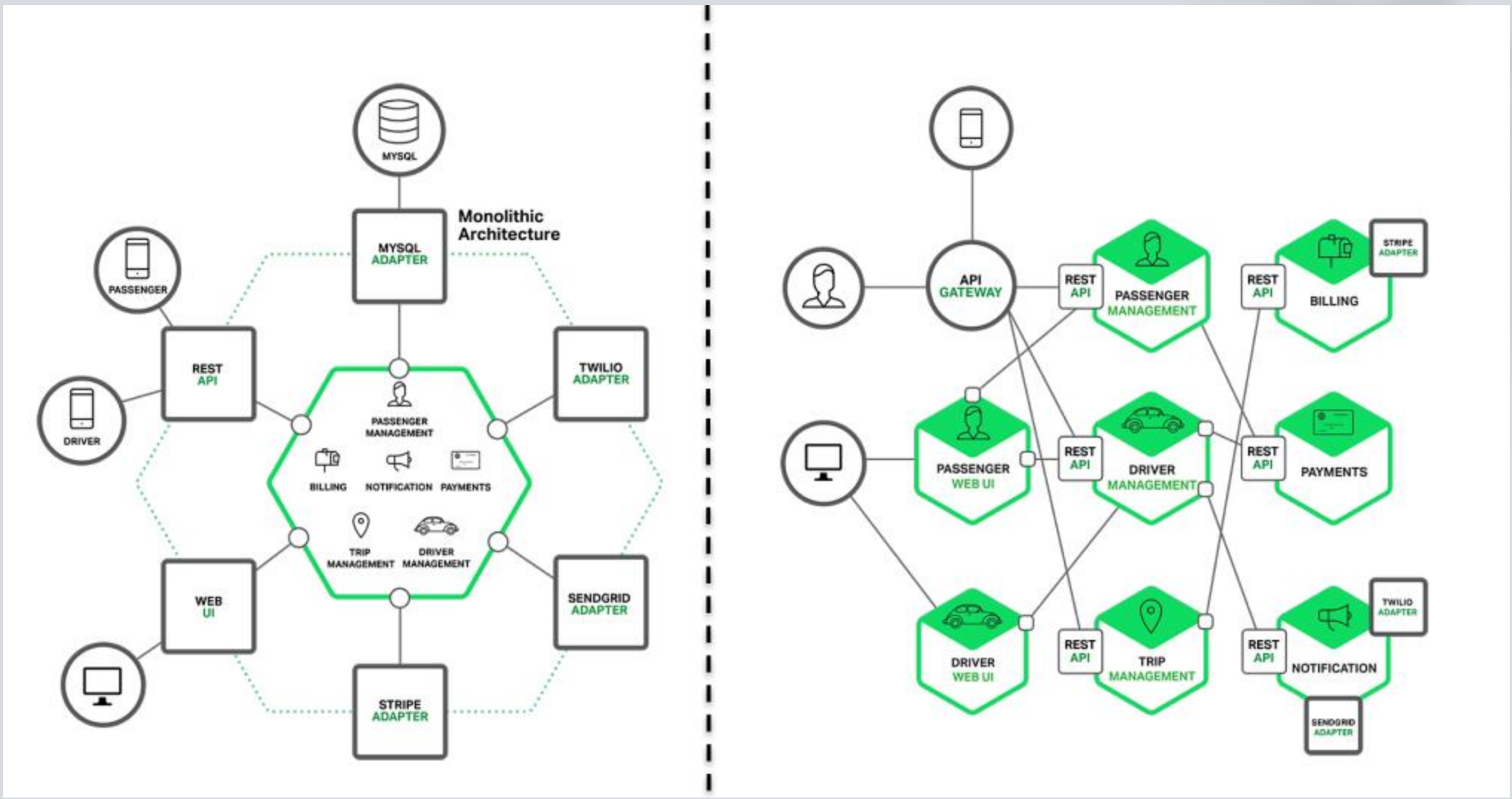
Microservizi

- L'architettura dei microservizi è un approccio progettuale per creare una singola applicazione come un set di **piccoli servizi**
- Ogni servizio viene eseguito nel proprio processo e comunica con altri servizi tramite un'interfaccia be definita, in genere un'interfaccia di programmazione delle applicazioni (**API**) basata su **HTTP**
- I microservizi sono costruiti attorno alle capacità aziendali: ogni servizio è limitato a un singolo scopo.



Punti Chiave

- è **piccolo** (“due settimane per riscriverlo da zero” [S.Newman 2015])
- ha una singola **responsabilità** (concern)
- rispecchia la **struttura** organizzativa dell’azienda (Conway’s law*, 1968)
- viene eseguito in un **processo** separato
- può utilizzare **framework** e linguaggi di programmazione eterogenei
- espone **interfacce** ben definite
- usa sistemi di **comunicazione** leggeri (message queue, REST API)
- può essere aggiornato e messo in **produzione** in maniera indipendente
- può **scalare** in maniera indipendente



Source: https://www.f5.com/company/blog/nginx/introduction-to-microservices

Infrastructure as a Code

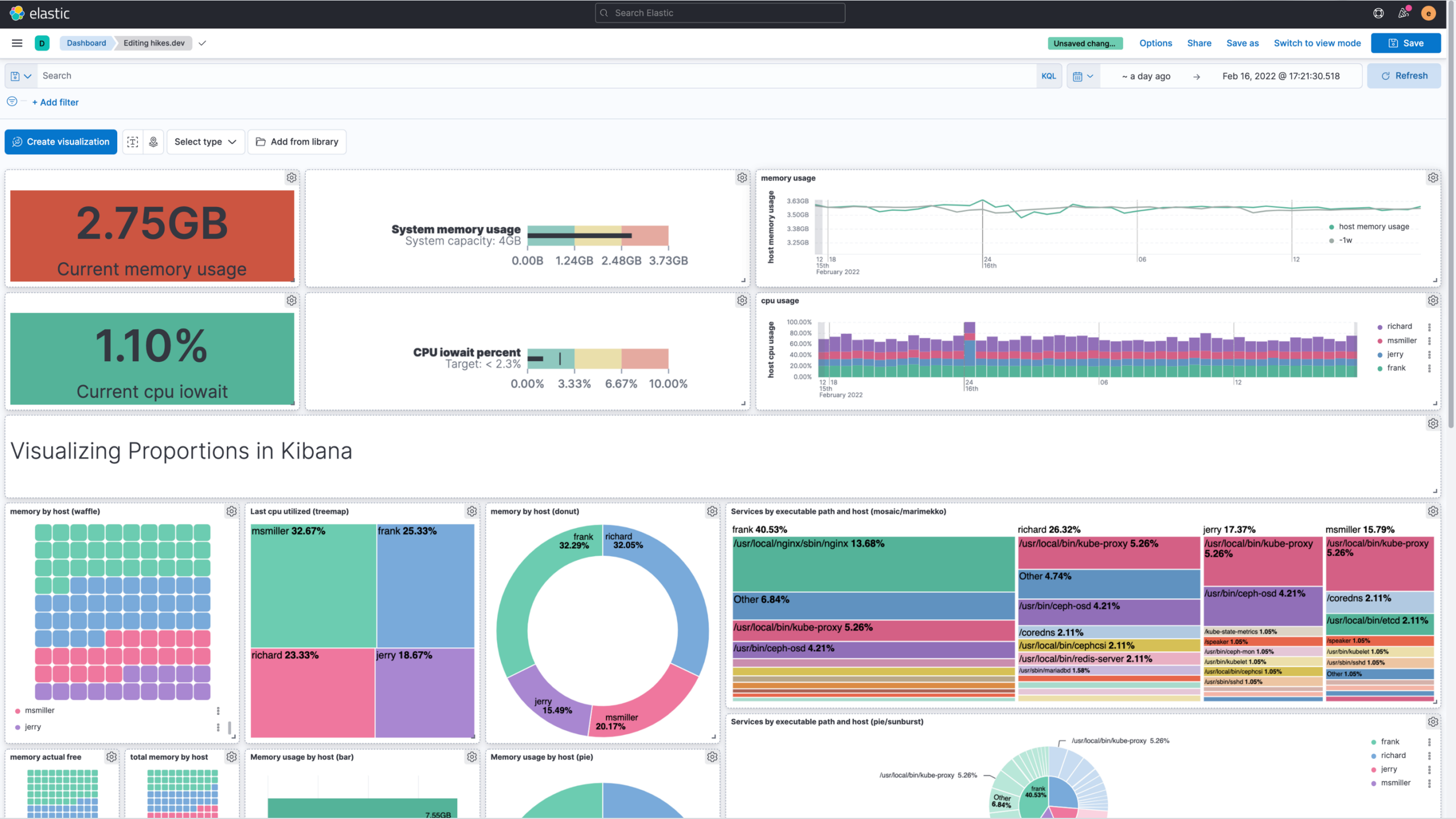
- E’ una pratica che consiste nel **gestire** e **configurare** l'infrastruttura IT utilizzando codice anziché configurazioni manuali o strumenti interattivi
- Con **IaC**, l'infrastruttura viene descritta attraverso file di configurazione, che possono essere gestiti e versionati allo stesso modo del software
- L'infrastruttura (server, reti, database, etc.) viene trattata come **software**, rendendo possibile automatizzare e riprodurre facilmente la creazione e gestione dell'intera infrastruttura.

Caratteristiche IaC

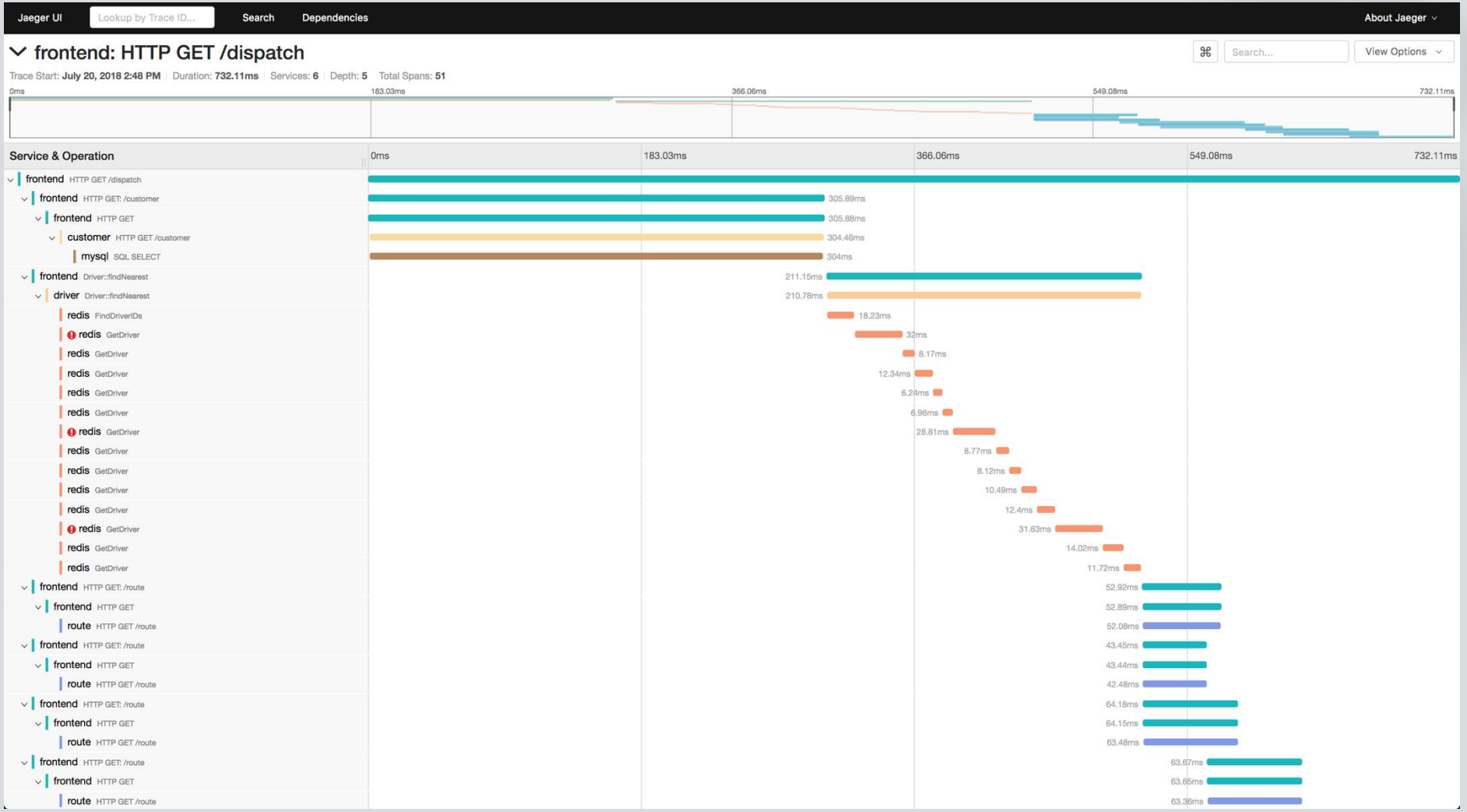
- **Automazione** - La configurazione dell'infrastruttura viene automatizzata tramite script e file di configurazione
- **Versioning** – Utilizzo di sistemi di versionamento (come Git) per tracciare le modifiche, eseguire rollback o comparare versioni precedenti delle configurazioni
- **Coerenza e riproducibilità** - ogni volta che si esegue il codice, si crea la stessa infrastruttura, eliminando i rischi di errori manuali e incoerenze tra gli ambienti di sviluppo, staging e produzione
- **Modelli dichiarativi o imperativi**
  - **Dichiarativi** Si definisce lo stato desiderato dell'infrastruttura, e gli strumenti IaC si occupano di far sì che l'infrastruttura arrivi a quello stato
  - **Imperativi** Viene specificato passo dopo passo come configurare l'infrastruttura

Monitoring and Logging

- Il **monitoraggio** riguarda la raccolta e l'analisi in tempo reale dei dati sulle prestazioni, sulla disponibilità e sull'uso delle risorse di un sistema
- Lo scopo del monitoraggio è quello di fornire **informazioni** dettagliate sul funzionamento delle applicazioni e dell'infrastruttura
  - Comprendere gli effetti delle **modifiche** introdotte nel codice
  - Comprendere le cause dei **problemi** e malfunzionamenti
- I dati raccolti vengono usati come **feedback** per capire quali modifiche introdurre nella prossima release


elastic
Search Elastic
Dashboard
Editing hikes.dev
Unsaved chang...
Options
Share
Save as
Switch to view mode
Save
Search
KQL
~ a day ago
Feb 16, 2022 @ 17:21:30.518
Refresh
+ Add filter
Create visualization
Select type
Add from library
2.75GB
Current memory usage
System memory usage
System capacity: 4GB
0.00B 1.24GB 2.48GB 3.73GB
memory usage
host memory usage
3.63GB
3.50GB
3.38GB
3.25GB
12 18 15th February 2022
24 16th
06
12
host memory usage
-1w
1.10%
Current cpu iowait
CPU iowait percent
Target: < 2.3%
0.00% 3.33% 6.67% 10.00%
cpu usage
host cpu usage
100.00%
80.00%
60.00%
40.00%
20.00%
0.00%
12 18 15th February 2022
24 16th
06
12
richard
msmiller
jerry
frank
Visualizing Proportions in Kibana
memory by host (waffle)
msmiller
jerry
Last cpu utilized (treemap)
msmiller 32.67%
frank 25.33%
richard 23.33%
jerry 18.67%
memory by host (donut)
frank 32.29%
richard 32.05%
jerry 15.49%
msmiller 20.17%
Services by executable path and host (mosaic/marimekko)
frank 40.53%
/usr/local/nginx/sbin/nginx 13.68%
Other 6.84%
/usr/local/bin/kube-proxy 5.26%
/usr/bin/ceph-osd 4.21%
richard 26.32%
/usr/local/bin/kube-proxy 5.26%
Other 4.74%
/usr/bin/ceph-osd 4.21%
/coredns 2.11%
/usr/local/bin/cephcsi 2.11%
/usr/local/bin/redis-server 2.11%
/usr/sbin/mariadbd 1.58%
jerry 17.37%
/usr/local/bin/kube-proxy 5.26%
/usr/bin/ceph-osd 4.21%
/kube-state-metrics 1.05%
/speaker 1.05%
/usr/bin/ceph-mon 1.05%
/usr/bin/kubelet 1.05%
/usr/local/bin/cephcsi 1.05%
/usr/sbin/sshd 1.05%
msmiller 15.79%
/usr/local/bin/kube-proxy 5.26%
/coredns 2.11%
/usr/local/bin/etcd 2.11%
/speaker 1.05%
/usr/bin/kubelet 1.05%
/usr/sbin/sshd 1.05%
Other 1.05%
Services by executable path and host (pie/sunburst)
/usr/local/bin/kube-proxy 5.26%
/usr/local/bin/kube-proxy 5.26%
Other 6.84%
frank 40.53%
frank
richard
jerry
msmiller
memory actual free
total memory by host
Memory usage by host (bar)
7.55GB
Memory usage by host (pie)

Monitoring su Microservizi



Communication and Collaboration

- L'automatizzazione del processo di delivery richiede una **collaborazione** stretta tra sviluppatori e sistemisti
- Necessario velocizzare la **comunicazione** (anche con altri dipartimenti, come marketing e vendite)
- Permettere a tutti i componenti dell'organizzazione di **allinearsi** più facilmente agli obiettivi del progetto
- Tool per la comunicazione (chat, messaggistica) e **condivisione** della conoscenza (versioning control system, pull request, issue tracker, wiki)

Università di Catania

Uni ct MATEMATICA E INFORMATICA

The End 

Alessandro Midolo, PhD

Dipartimento di Matematica e Informatica

Università di Catania

alessandro.midolo@unict.it

https://www.dmi.unict.it/amidolo/

A.A. 2024/2025