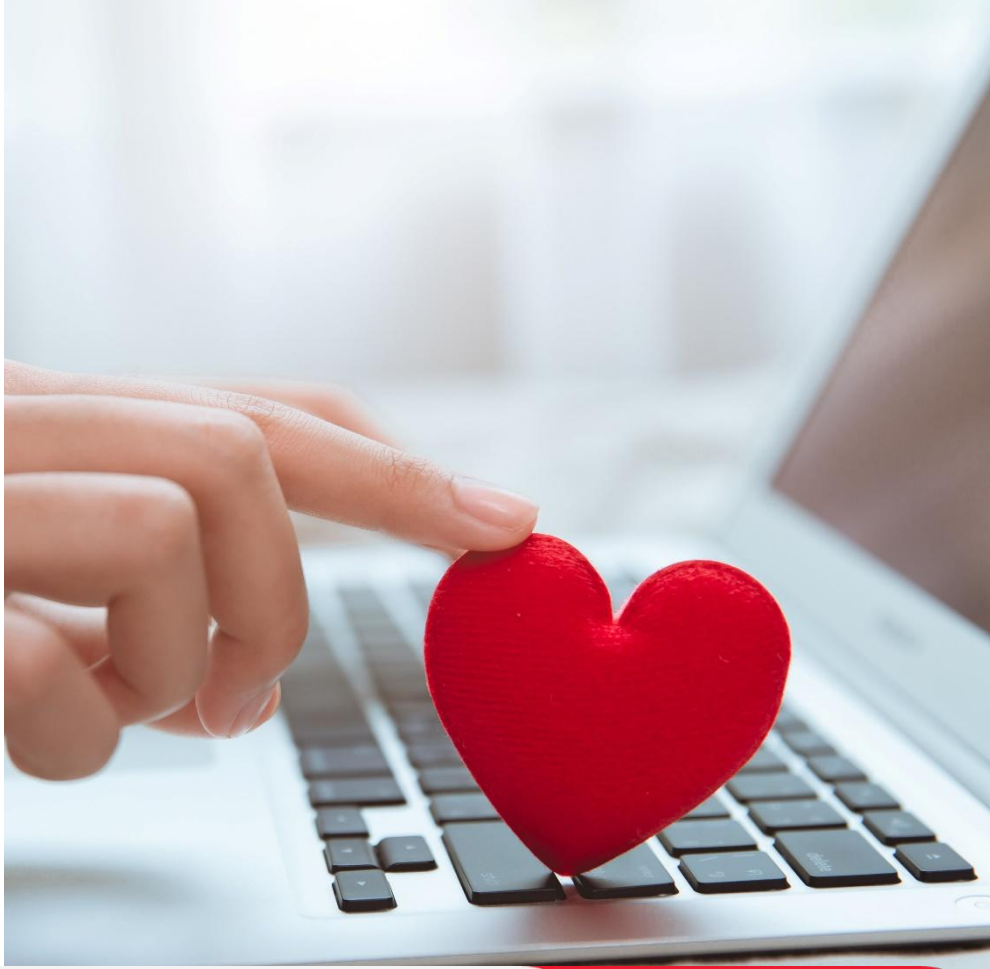
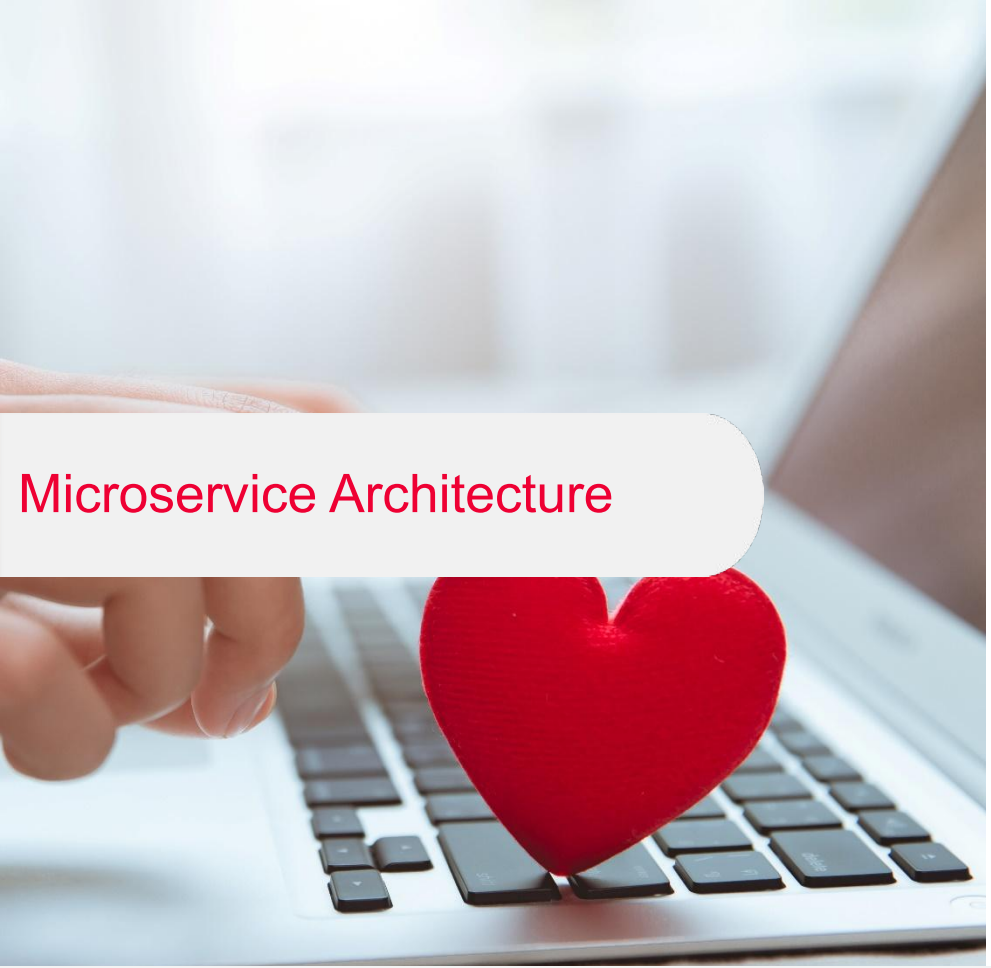
VTNET Cloud Team

VIETTEL MICROSERVICES ARCHITECTURE

Session Topics

- Microservice Architecture
- Cloud-Native application
- CI/CD
- Monitoring & Observability
- Case study: Viettel software development



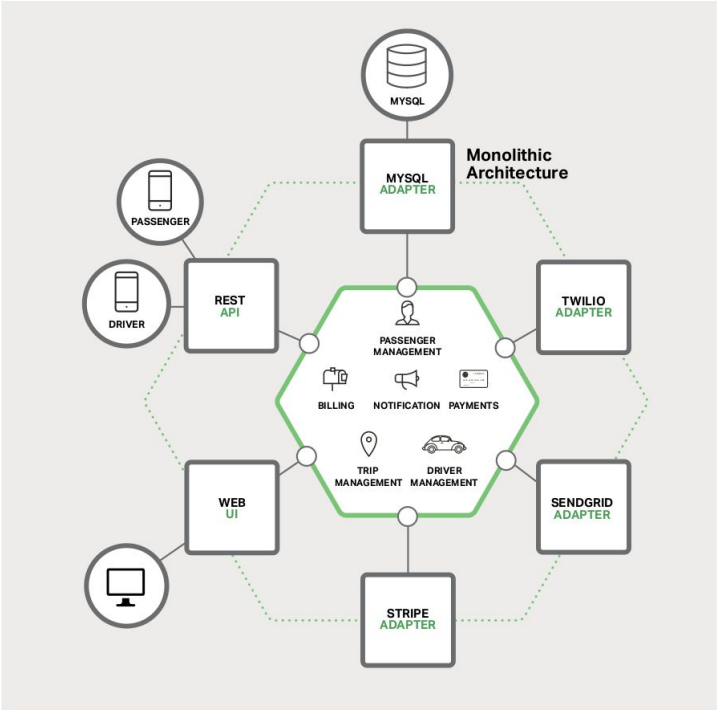


Microservice Architecture

Write “Well Code” is hard

- Paradigm, philosophy: OOP, SOLID, DRY, KISS, YAGNI...
- Design Pattern:
  - MVC
  - Dependency Injection
  - Hexagonal Architecture
- Auto tests: Unit, Component, Acceptance...

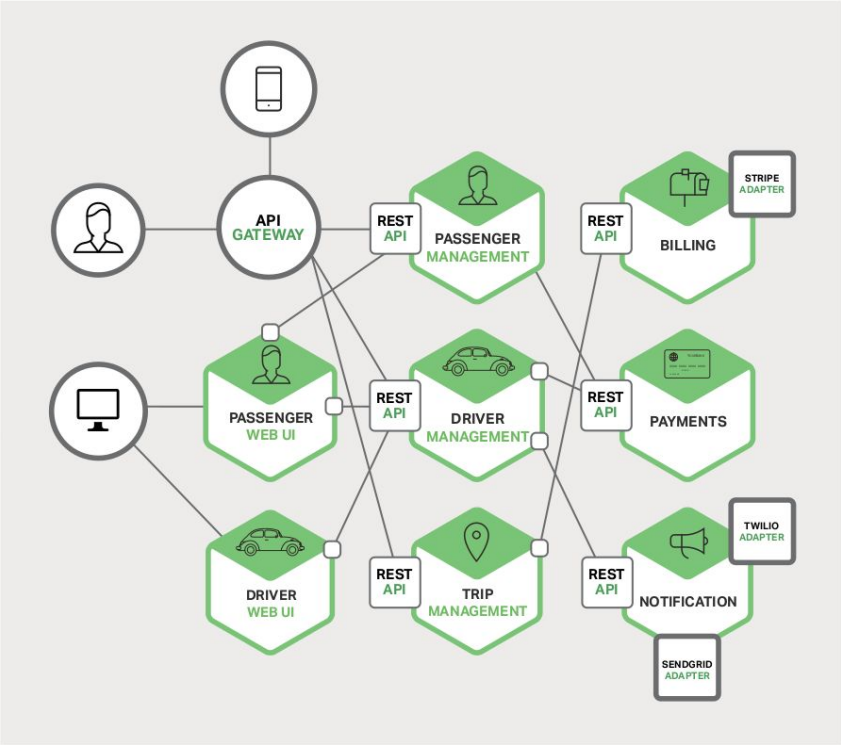
But over time, “Well Code” system growing



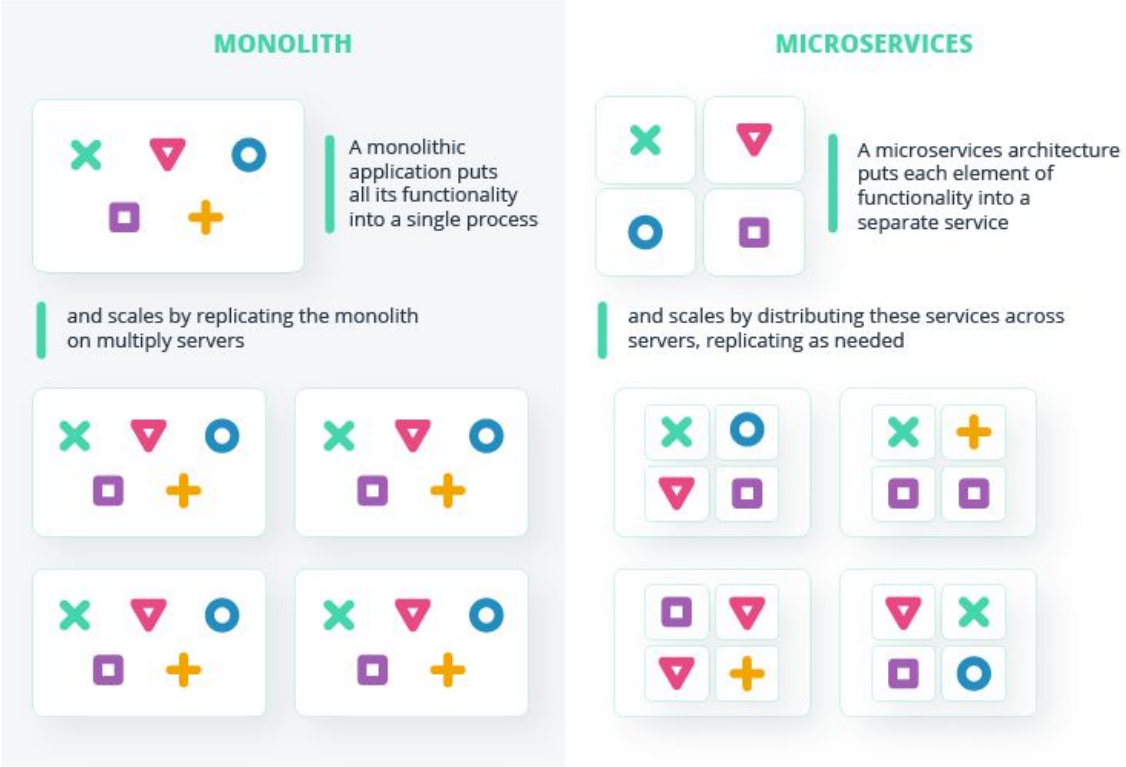
Problem when system growth

- Large code base, multiple functions in one repository.
- Too many dependencies need to managed.
- Too many tests, need many hours to complete tests.
- Hard to scale system.
- Hard to adopt new technologies
- Big ball of mud

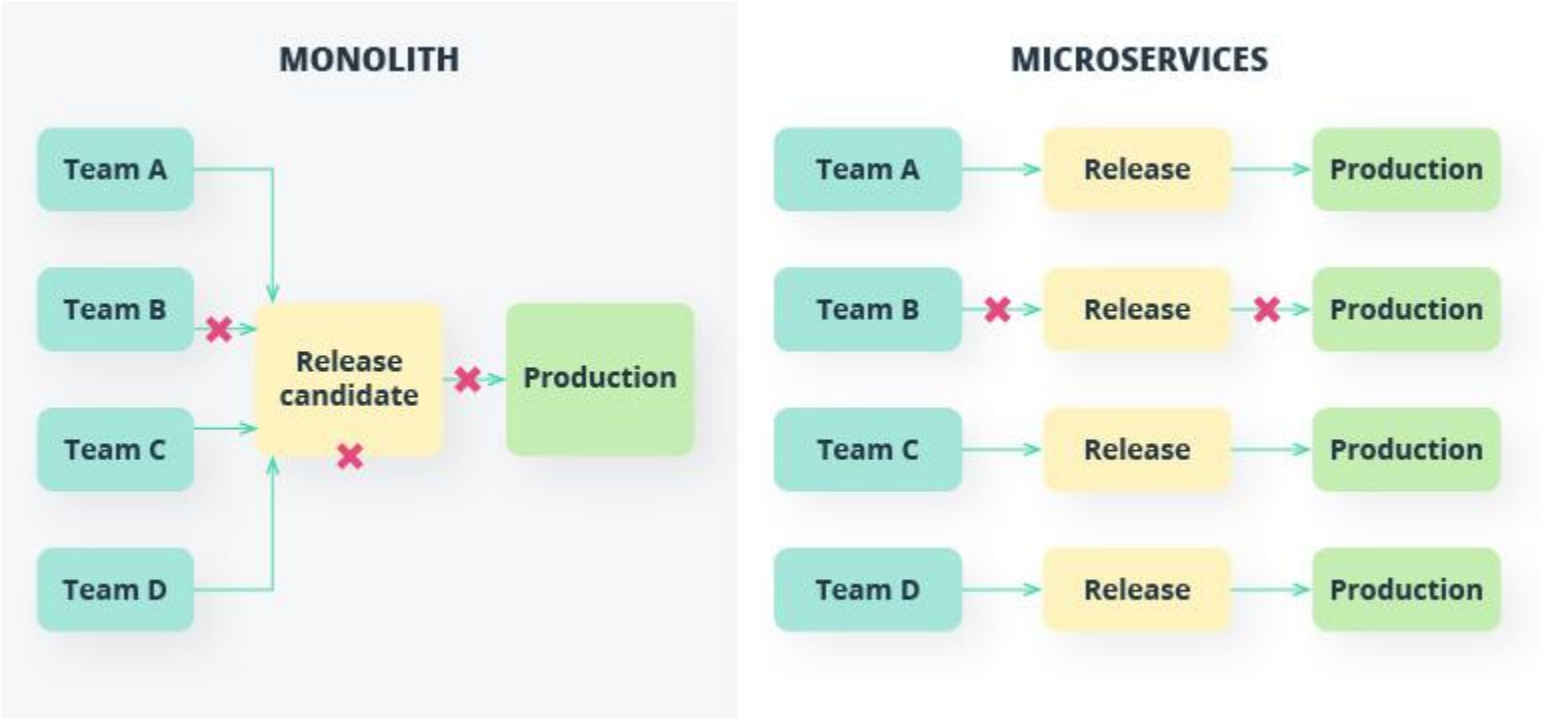
A solution for big systems: Microservice



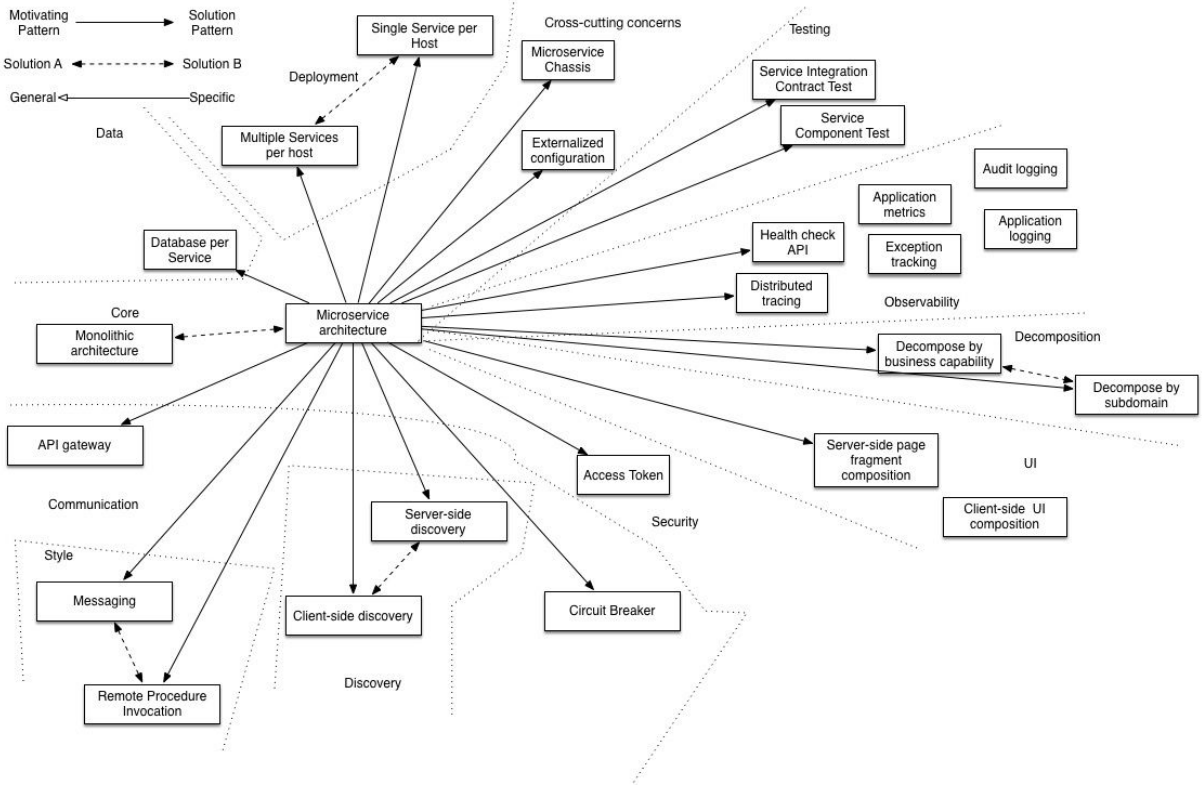
Microservice vs Monolith

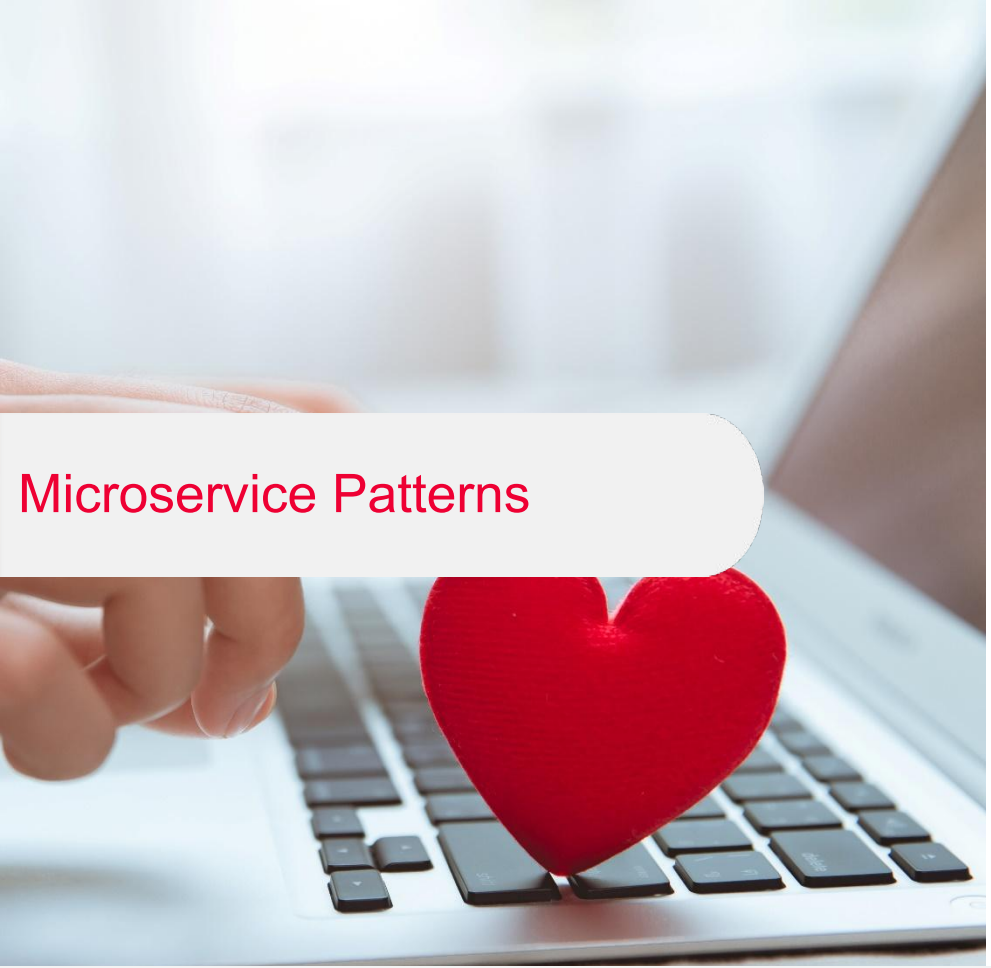


Microservice vs Monolith



Microservice Problems And Patterns



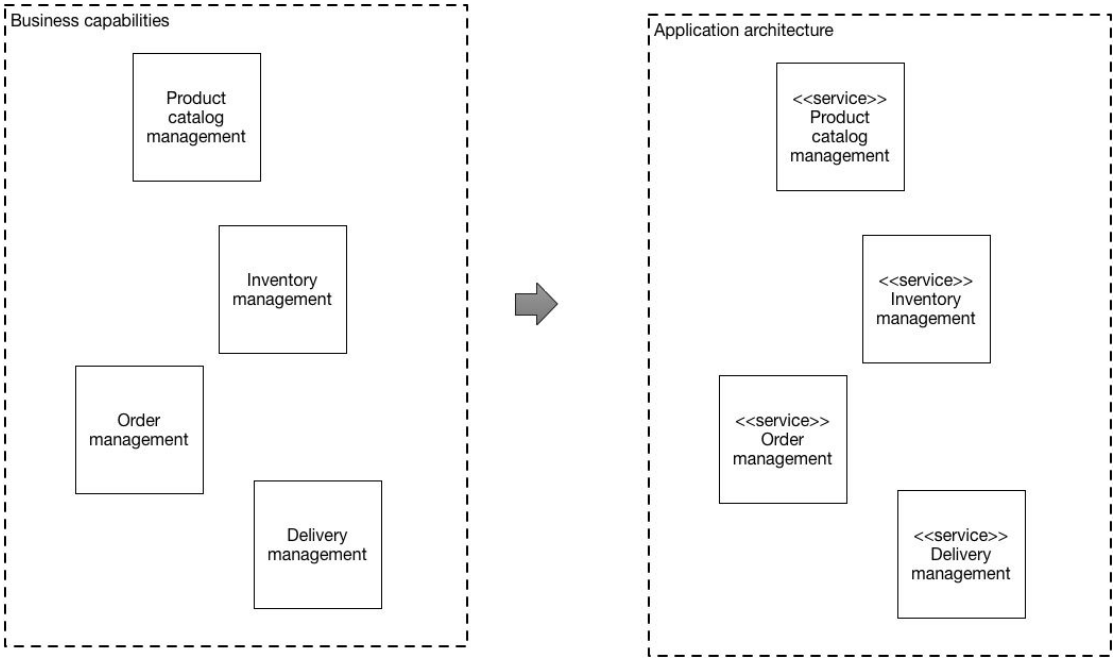


Microservice Patterns

- Decomposition
- Database Management
- Communication
- Deployment
- Observability

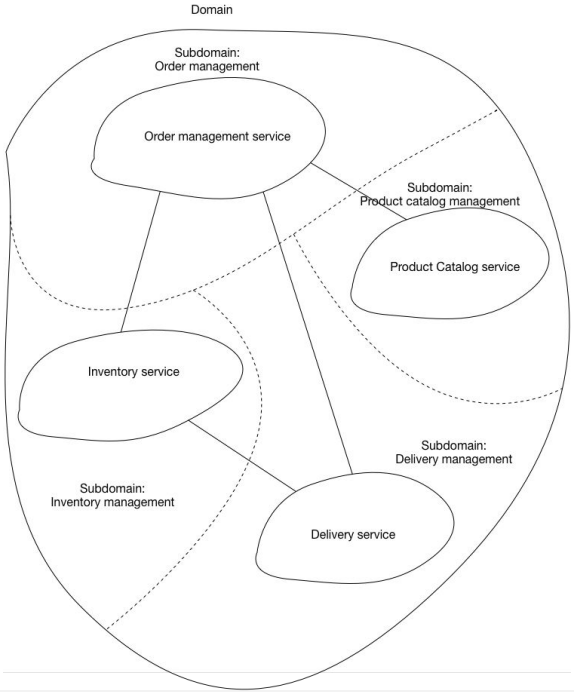
Decomposition

Decompose by business capability



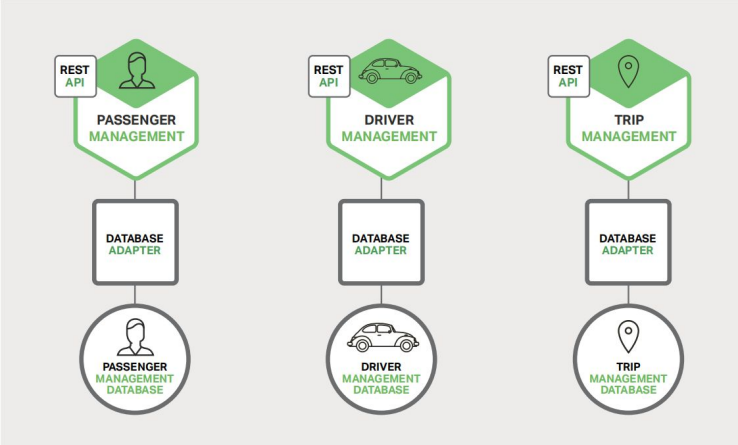
Decomposition

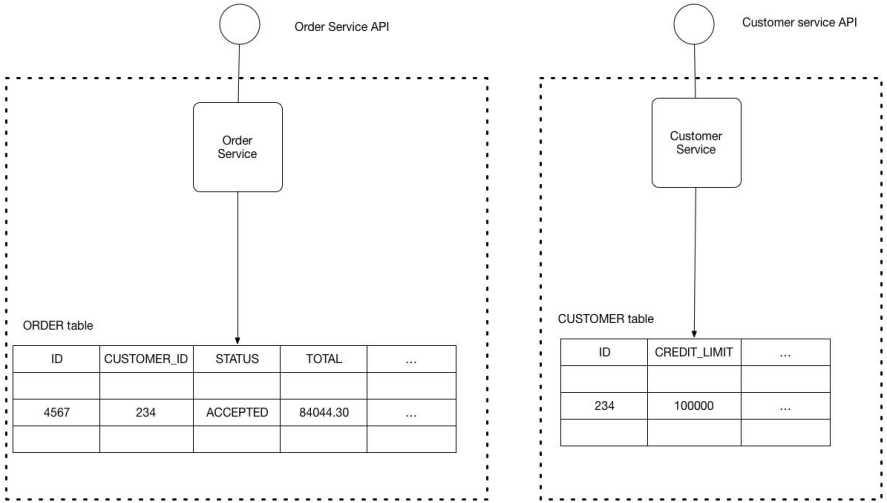
Decompose by subdomain



Database Management

- Private-tables-per-service
- Schema-per-service
- Database-server-per-service



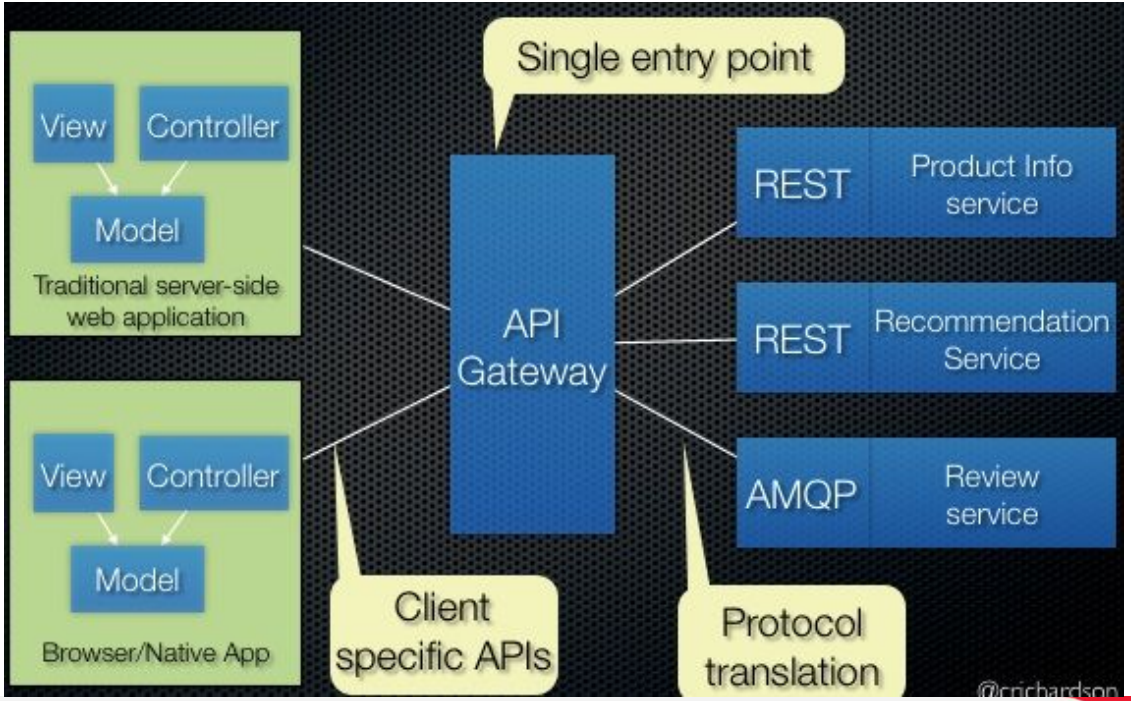


Communication

- External API
  - API Gateway
  - Backend for front-end
- Service-to-service communication
  - Hub communication
  - Client-side discovery
  - Server-side discovery
- Circuit Breaker

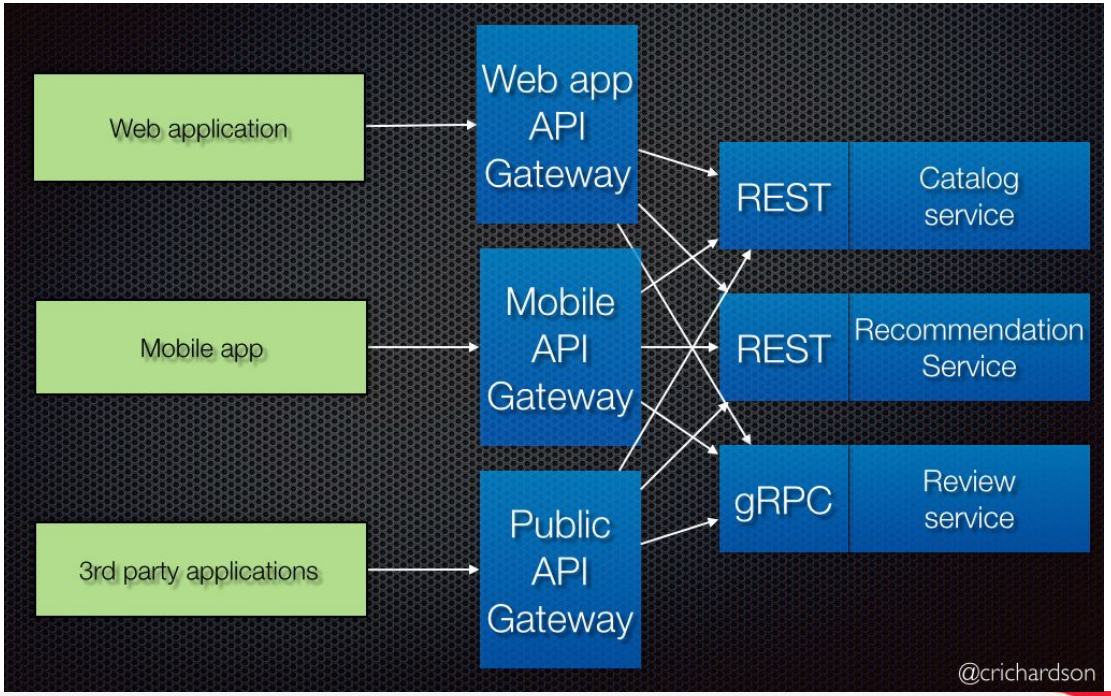
External API

API Gateway pattern

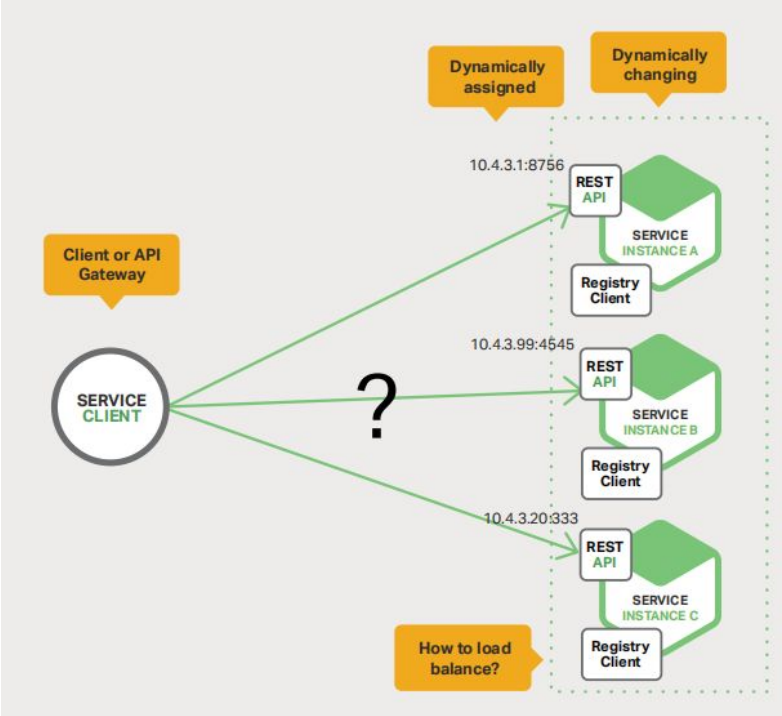


External API

Variation: Backends for frontends

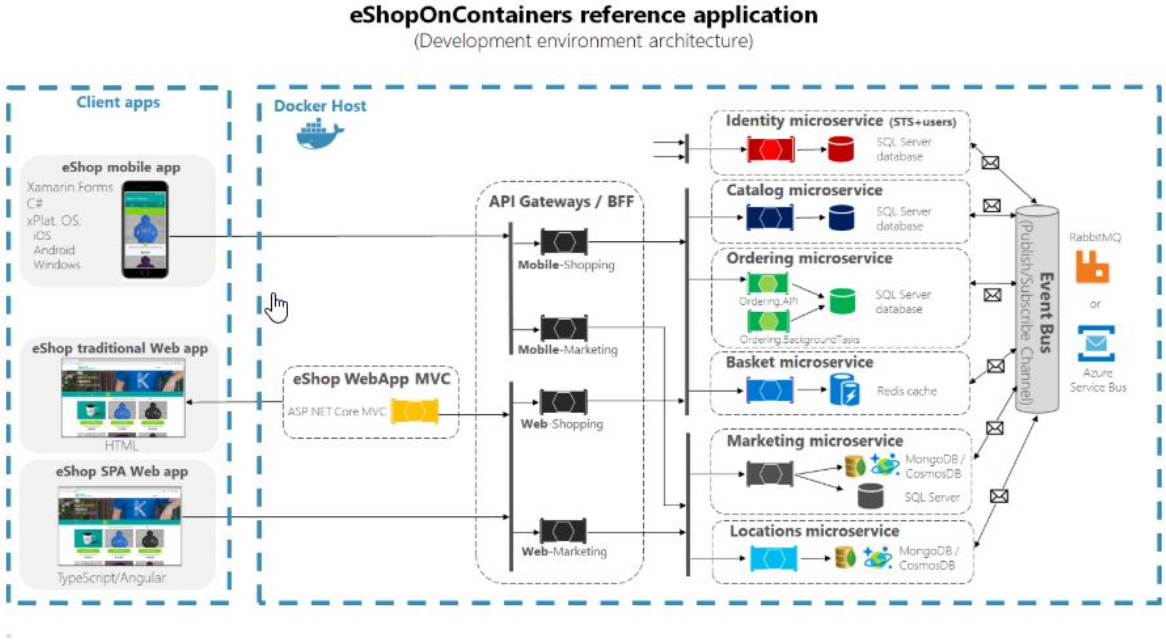


Service-to-service communication



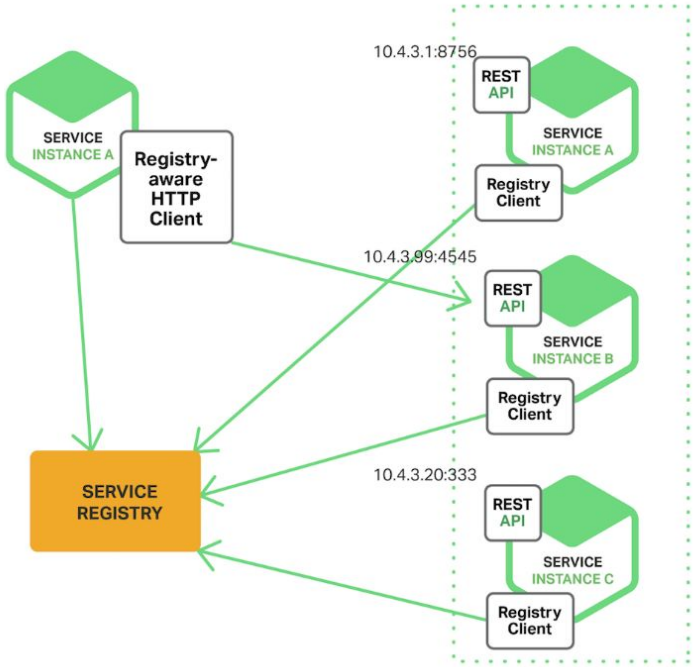
Service-to-service communication

Hub communication



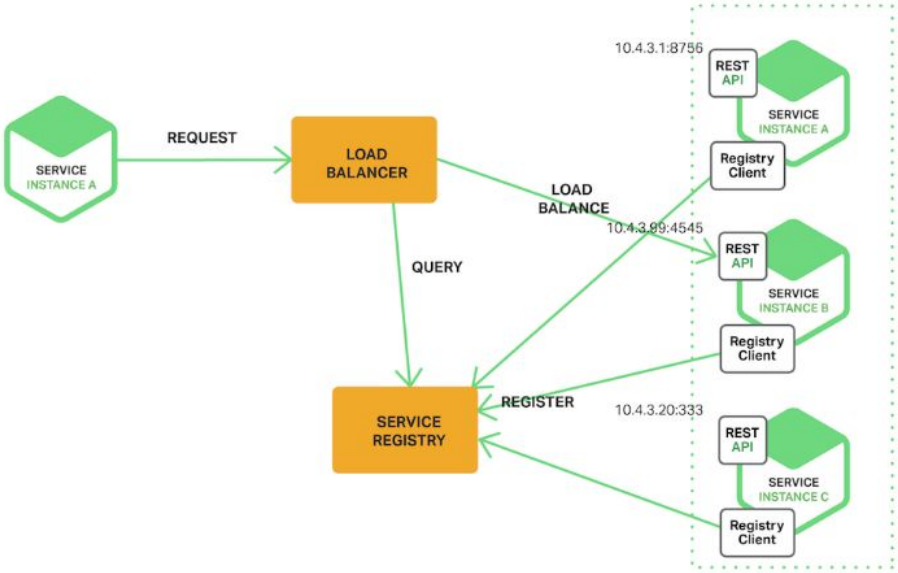
Service-to-service communication

Client-side discovery



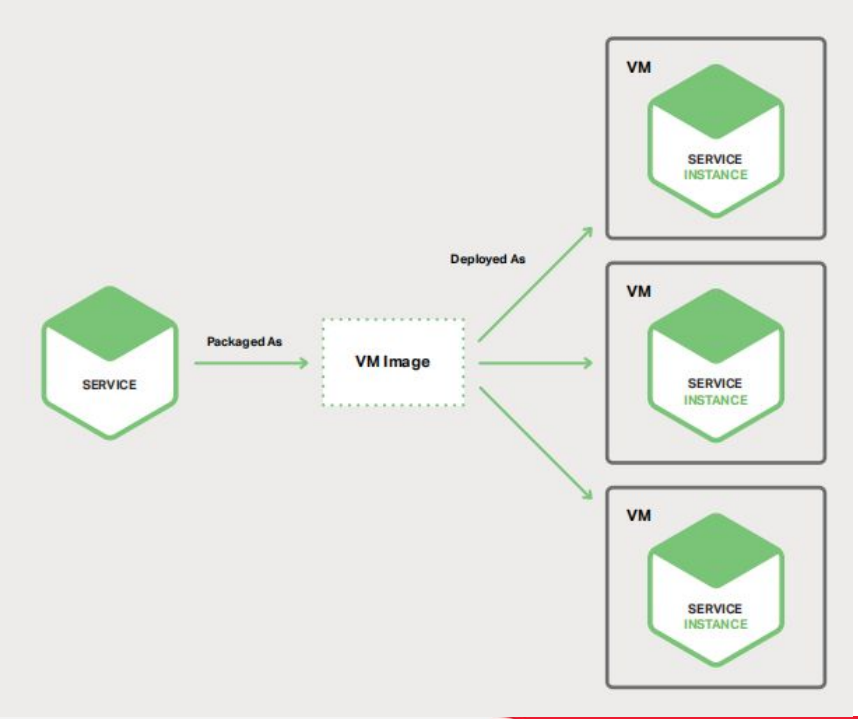
Service-to-service communication

Server-side discovery



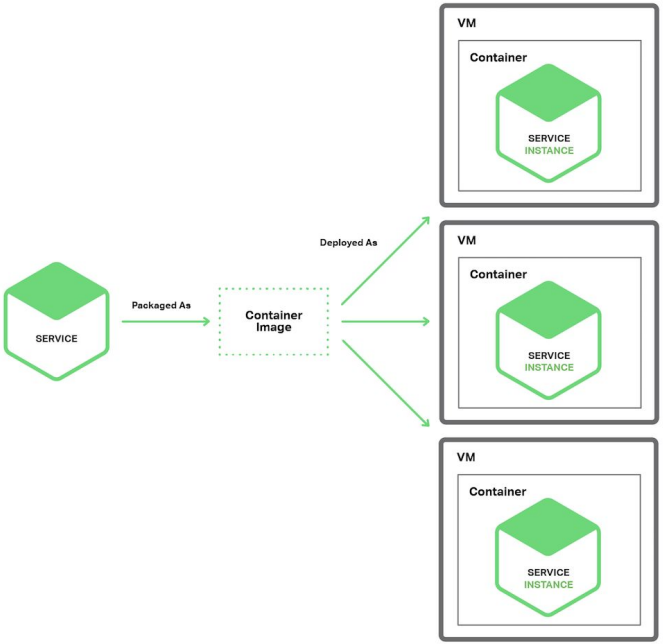
Microservice Deployment

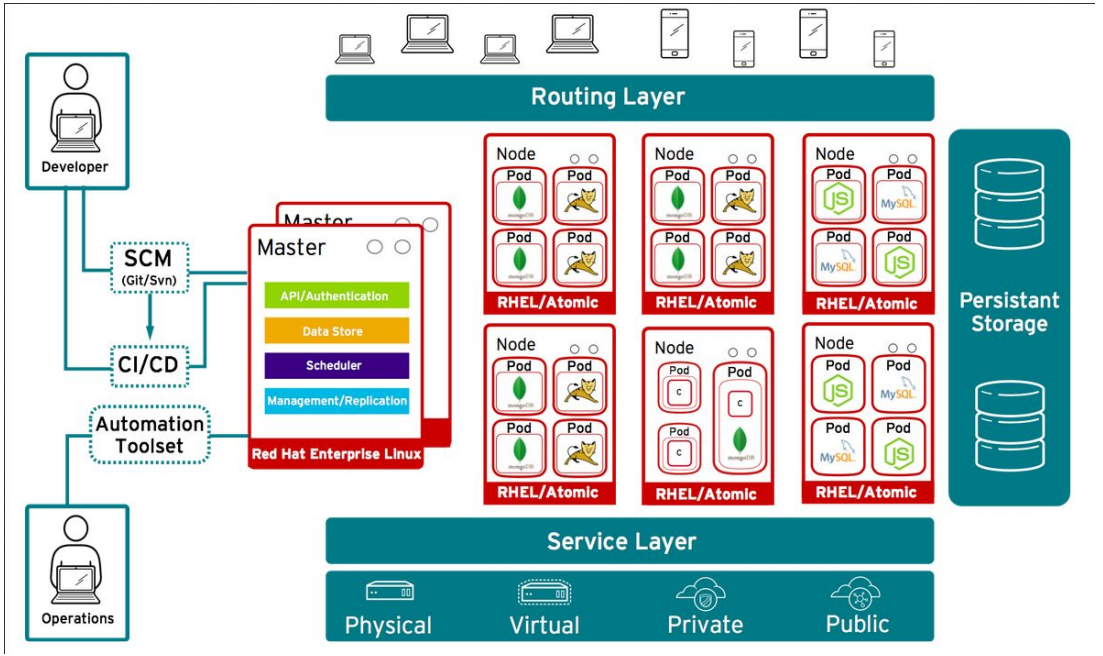
Service Instance per Virtual Machine Pattern



Microservice Deployment

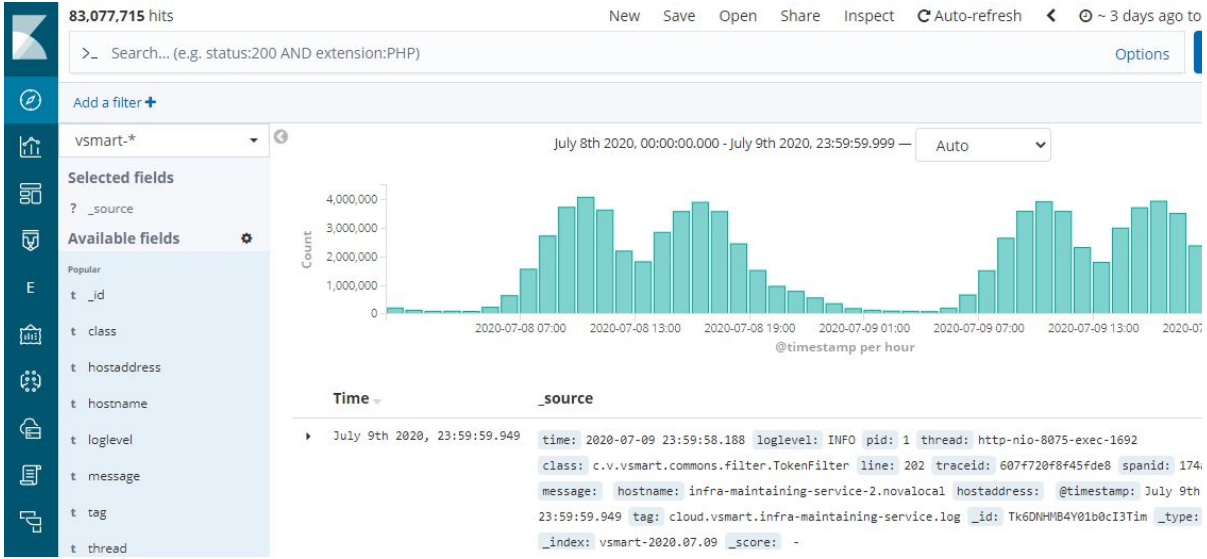
Service Instance per Container Pattern





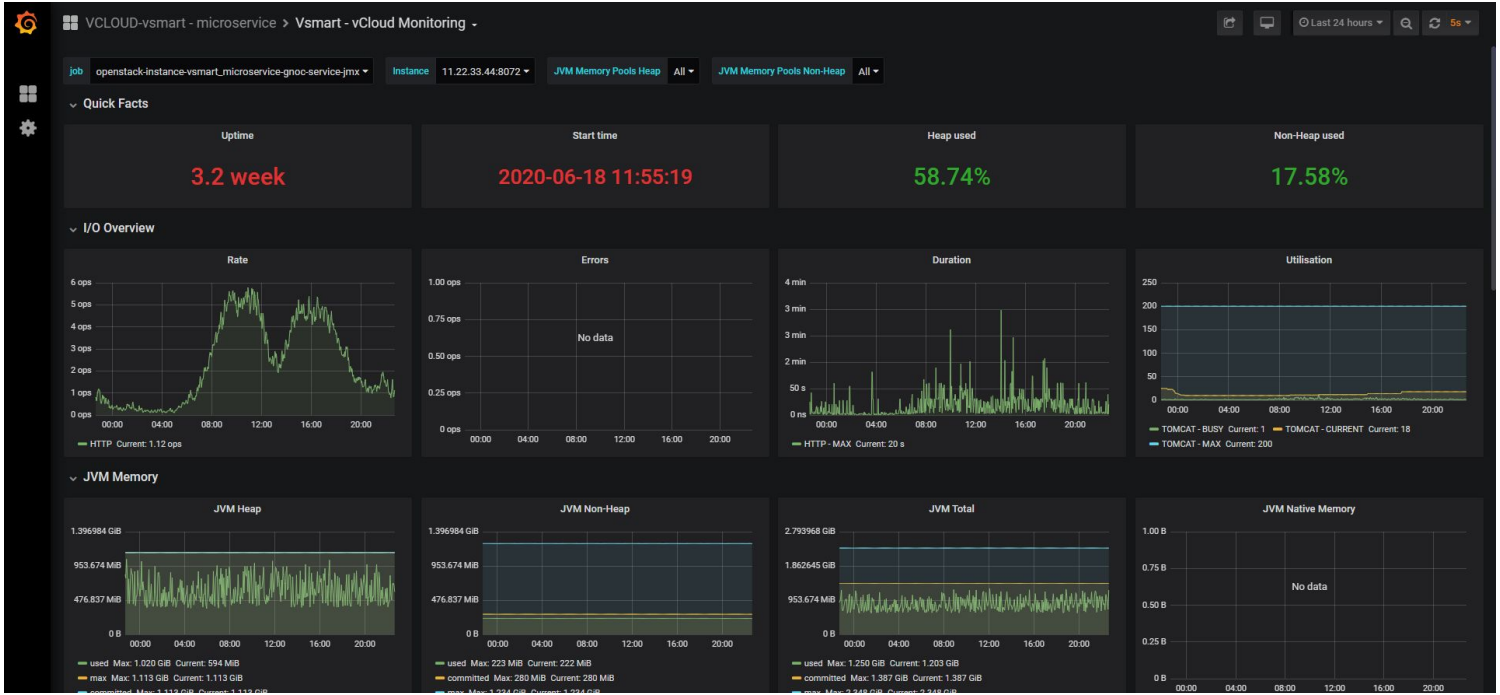
Observability

Logging



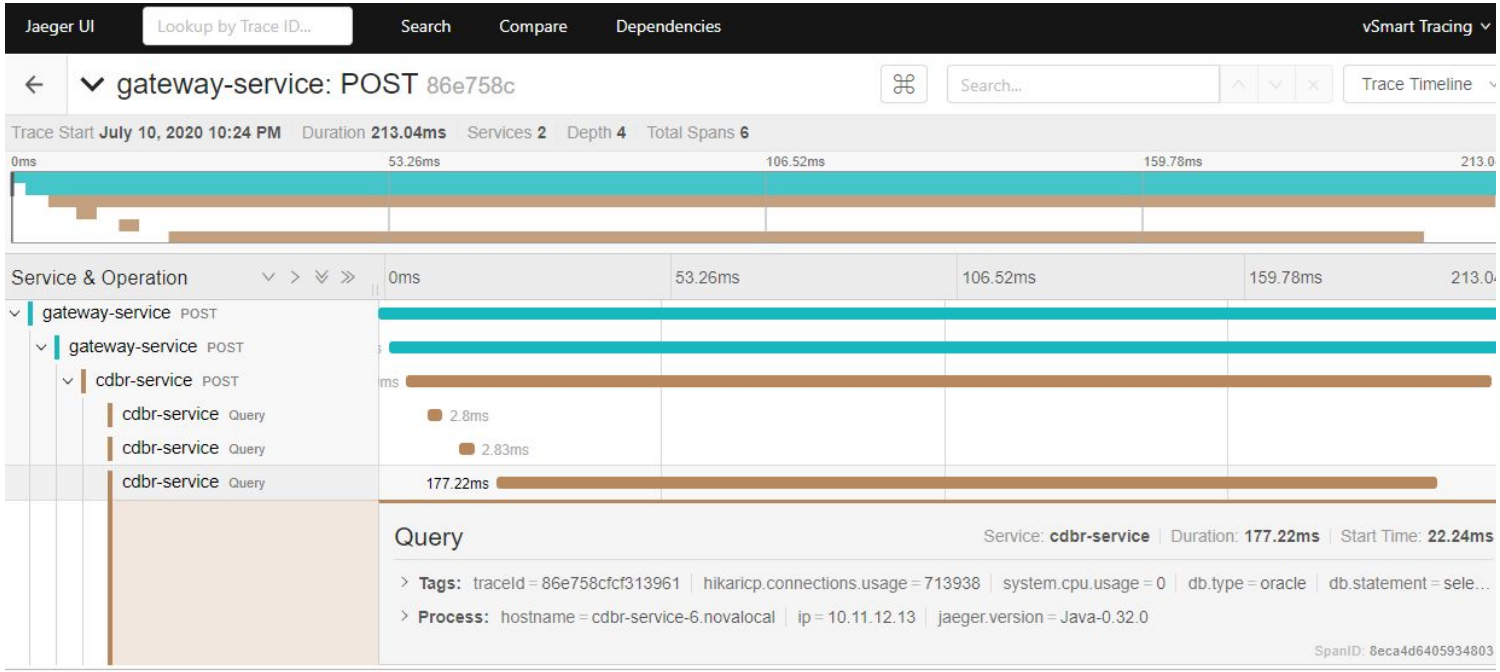
Observability

Monitoring



Observability

Tracing



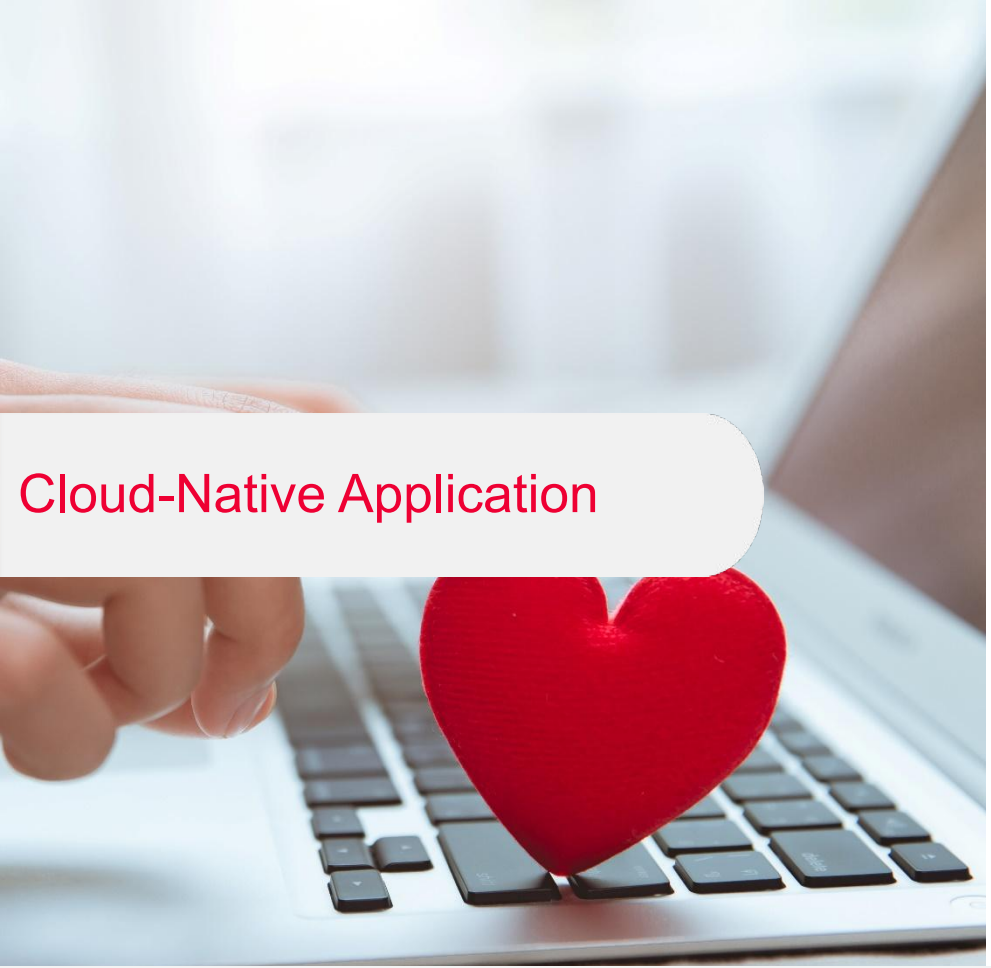
Microservices Anti-Patterns

Microservices Anti-Patterns

- Trying to fly before you can walk
- Scattershot adoption
- Distributed monolith
- Focussing on Technology
- No API semantic versioning
- Shared Libraries
- Mesh dependencies

How to avoid them?

- Learn to write good monolith before go microservices
- Clear design, single responsible separation
- Teams coordination and governance
- Devops, CI/CD cultural



Cloud-Native Application

- Introduction
- Characteris
- Benefits
- 12-factors application methodology

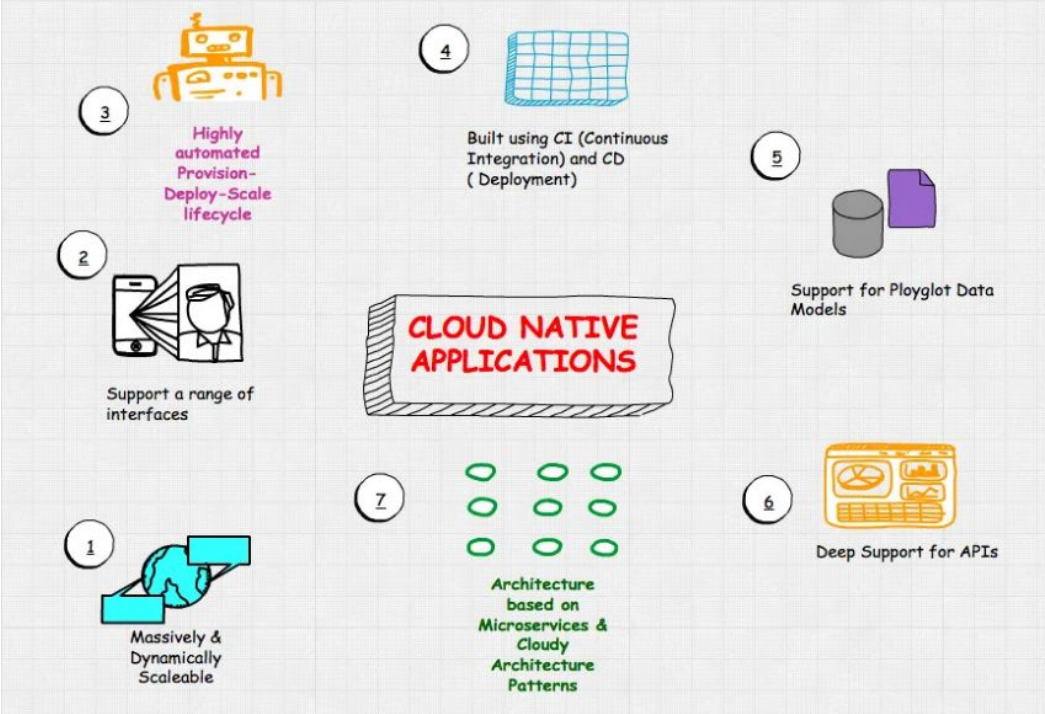
Cloud-Native - Definition

An approach in software development, to:

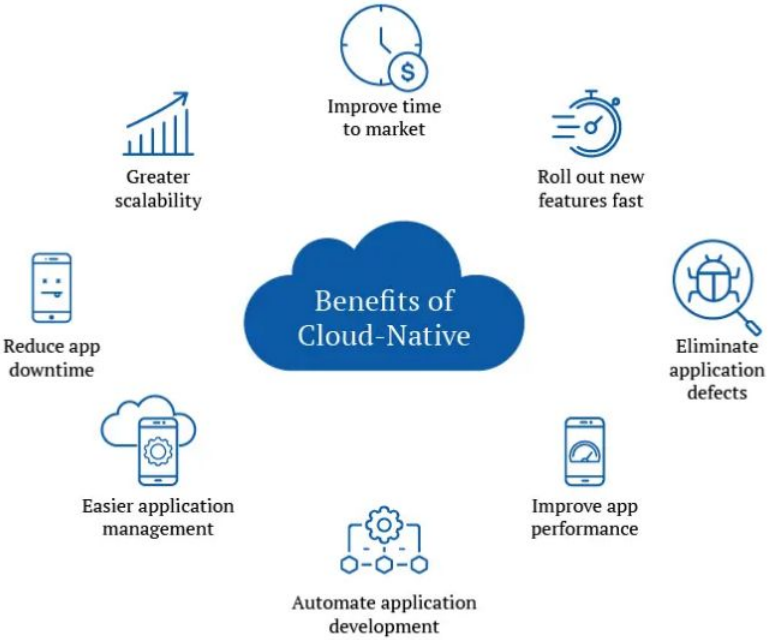
“build and run scalable applications in modern, dynamic environments such as public, private, and hybrid clouds”

https://github.com/cncf/toc/blob/main/DEFINITION.md

Cloud-Native - Application Characteris



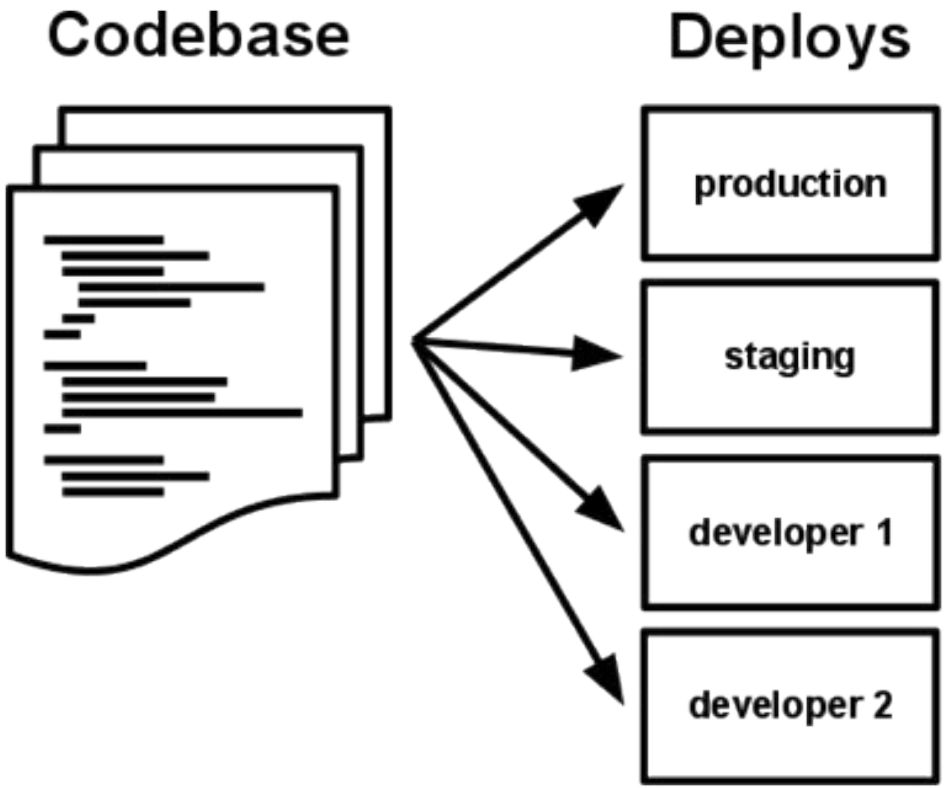
Cloud-Native - Benefits

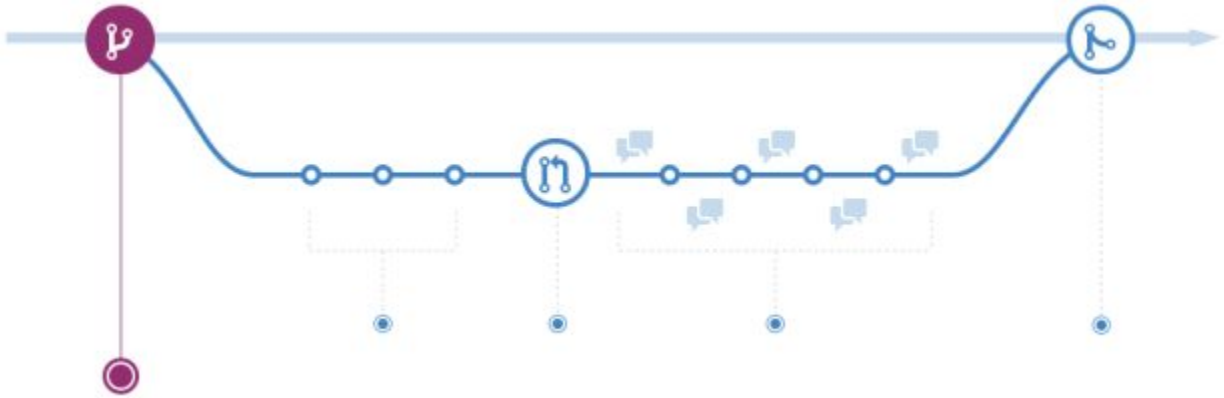


Cloud-Native - Methodology: 12 Factors

1
Codebase

One codebase tracked in revision control, many deploys

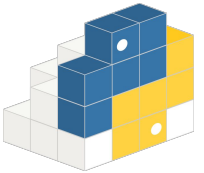




GitHub flow

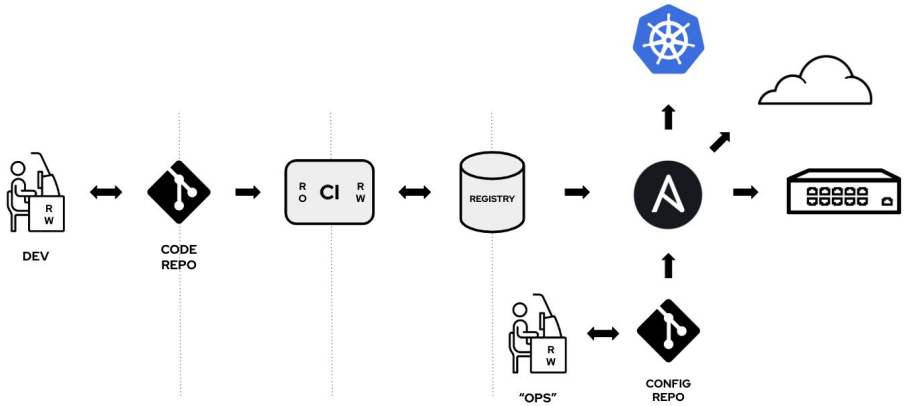
2
Dependencies

Explicitly declare and isolate dependencies



3
Config

Store config in the environment



In java source code (a lot):

```
service = new VsaadminServiceService(url,
     new QName("http://service.vsaadmin.viettel.com/", "VsaadminServiceService")
);
```

In xml file (a lot):

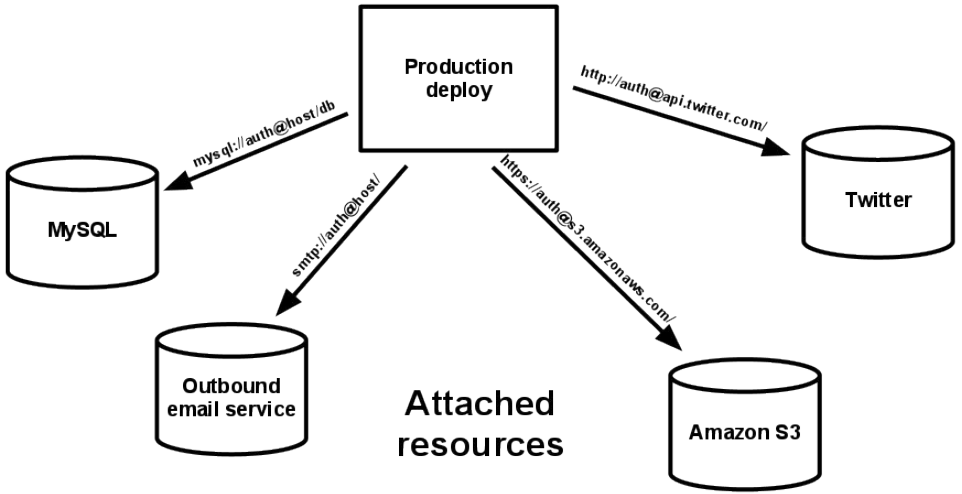
```
<property name="hibernate.connection.url">jdbc:oracle:thin:@11.22.33.44:1234:PM1</property>
<property name="hibernate.connection.username">qlctkt</property>
<property name="hibernate.connection.password">qlctkt@superpassw0rd</property>
```

Username and password were check-in VCS many times

Traditional application config styles

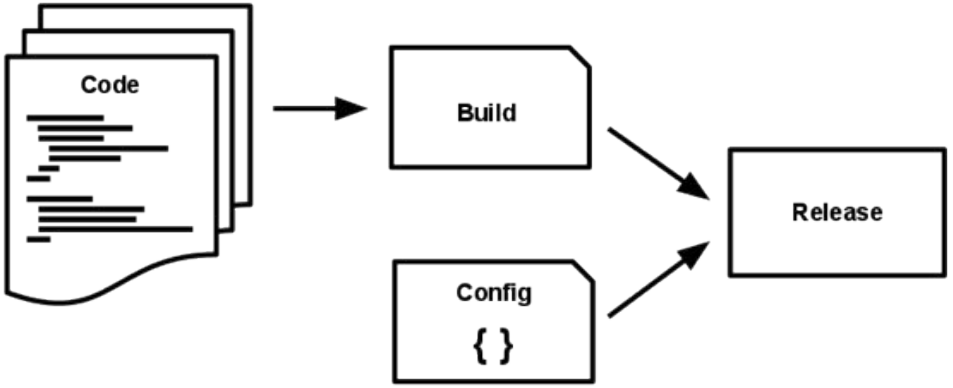
4
Backing services

Treat backing services as attached resources



5
Build, release, run

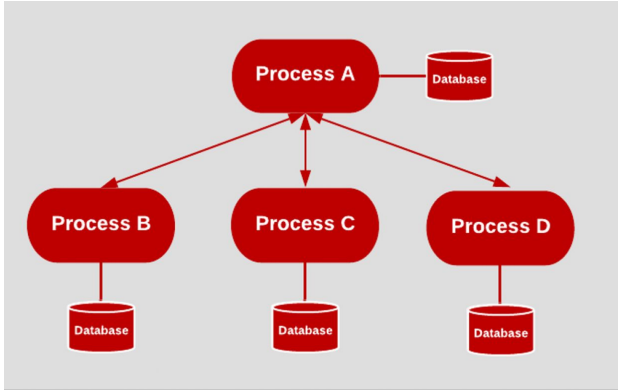
Strictly separate build and run stages



6
Processes

Execute the app as one or more **stateless** processes

- Stateless & shared-nothing
- Persistent data is stored in stateful backing services



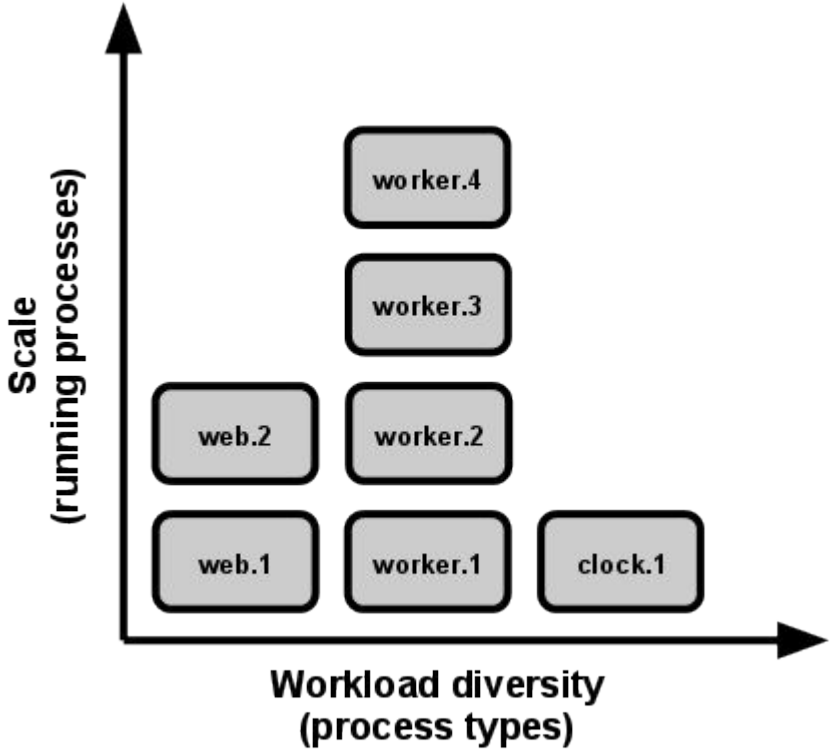
7
Port binding

Export services via port binding

- Self-contained
- Expose URI
- One app can become the backing service for another app

8
Concurrency

Scale out via the process model



9
Disposability

Maximize robustness
with **fast startup**
and **graceful shutdown**

- Mutable configuration
- Auto restart
- Live reload for web apps
- Crash-only design ☺️

10
Dev/prod parity

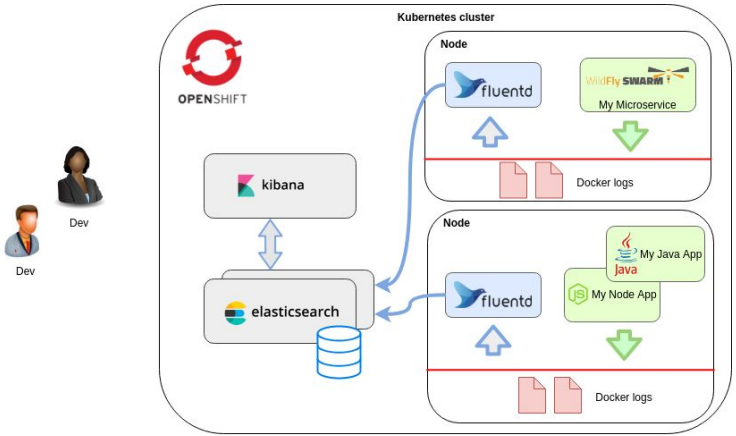
Keep development, staging, and production
as similar as possible

- Small changes
  (large ➜ focused small part)
- Reduce time gaps
  (weeks ➜ hours)
- CI/CD shining here
- Tools: docker, vagrant, ansible, chef, puppet, saltstack

11
Logging

Treat logs as **event streams**

- ~~*Log file*~~ → Stream
- Log routers: fluentd, logstash
- Log destination:
  stdout, *log file*, log storages,
  elasticsearch



12
Admin process

Run admin/management tasks as one-off processes

- Types: db migration, console, commit script
- Migration: Flyway, Liquibase, alembic, Django ORM
- Run in identical env, same on any release
- Ship with app code to avoid synchronization issues

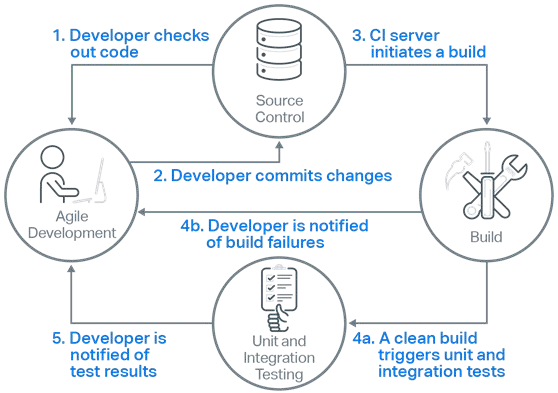
CI/CD

- Continuous Integration
- Continuous Delivery
- CI/CD Tools And Systems

Continuous Integration Definition

“Continuous Integration (CI) is a software development practice where members of a team integrate their work frequently, usually each person integrates at least daily - leading to multiple integrations per day. Each integration is verified by an automated build (including test) to detect integration errors as quickly as possible. Many teams find that this approach leads to significantly reduced integration problems and allows a team to develop cohesive software more rapidly”

Martin Fowler



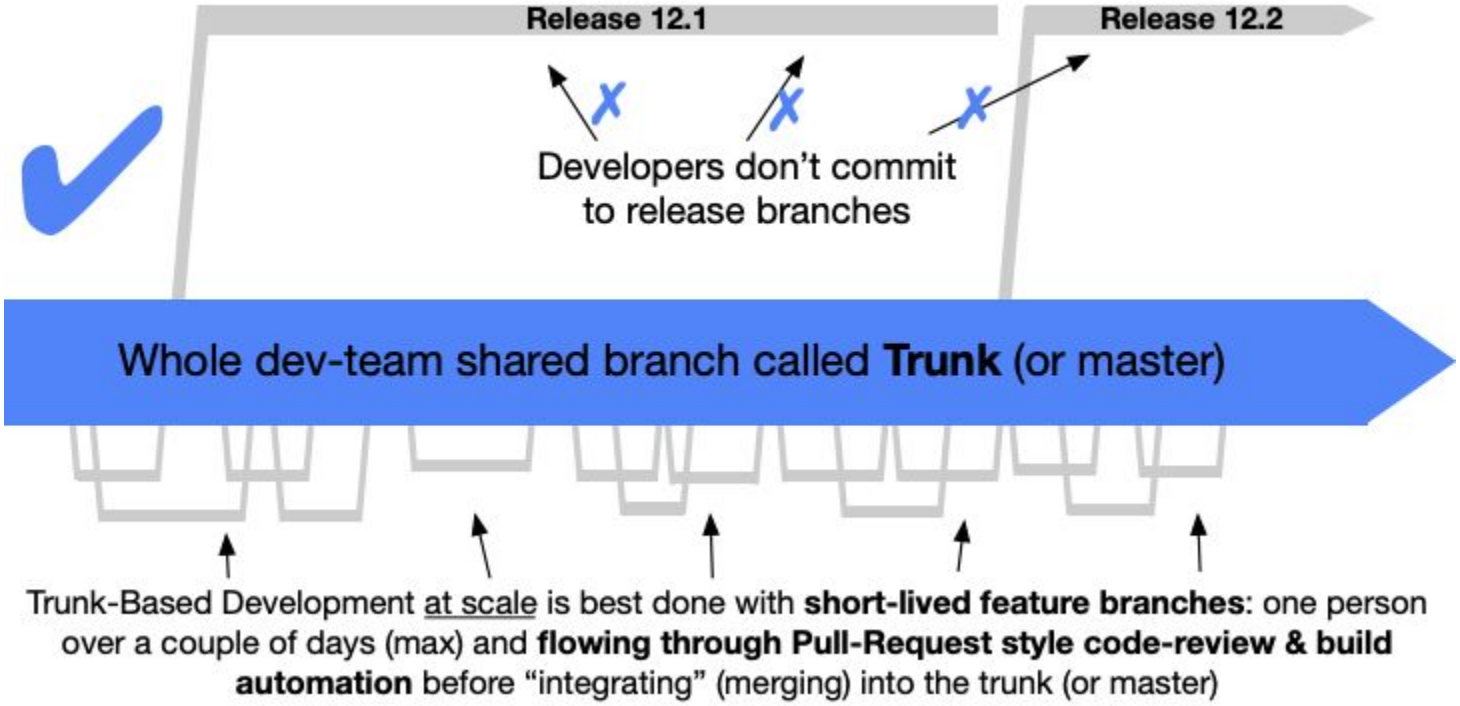
CI Benefits

- Avoid “integration hell”
- Fast feedback loops
- Detect and fix issues early
- Improve quality and testability

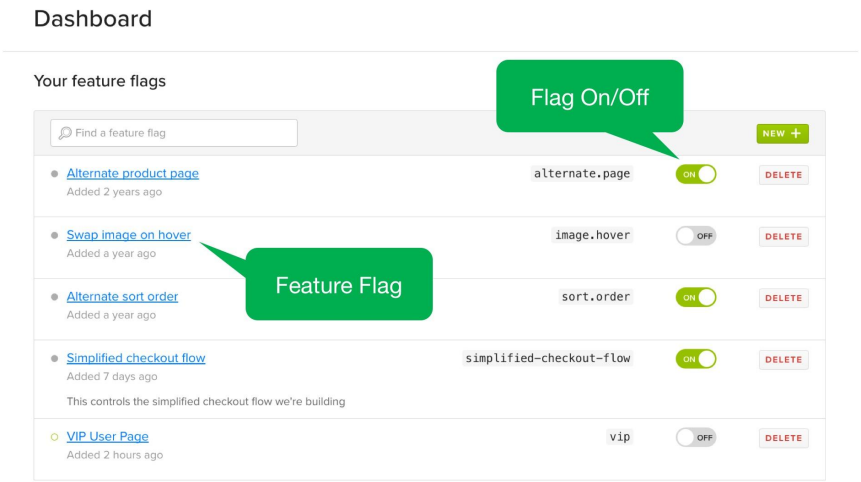
CI Techniques

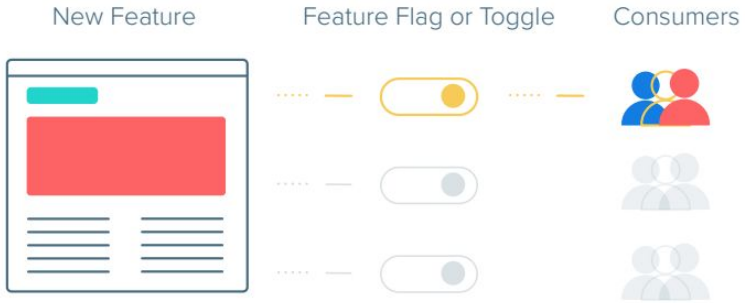
- Maintain a Single Source Repository
- Small Batches of code in a integration
- Automate the Build & Test
- Make Your Build Self-Testing
- Every Commit Should Build the Mainline on an Integration Machine
- Fix Broken Builds Immediately
- Keep the Build & Test Fast
- Test in a Clone of the Production Environment
- Everyone can see what's happening

CI Methodologies - Trunk Based Development



CI Methodologies - Feature Flags

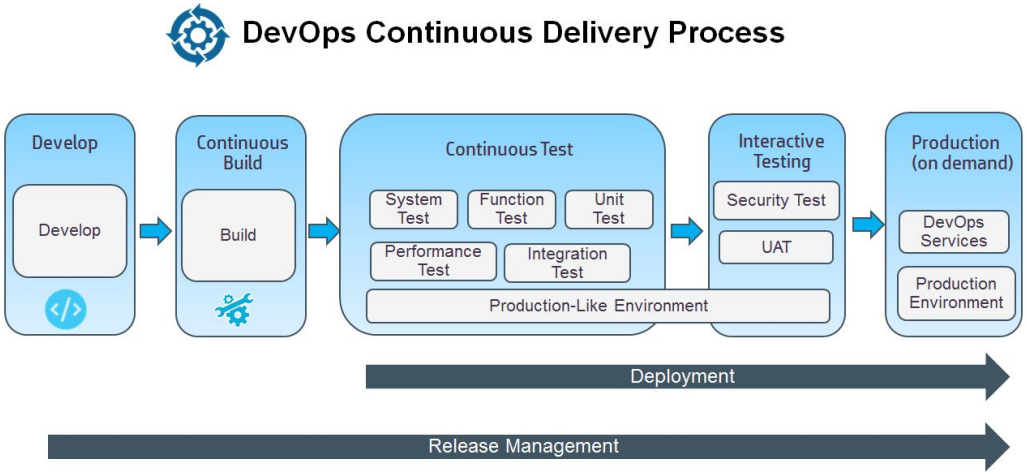




Continuous Delivery Definition

“Continuous Delivery is the ability to get changes of all types—including new features, configuration changes, bug fixes and experiments—into production, or into the hands of users, safely and quickly in a sustainable way.”

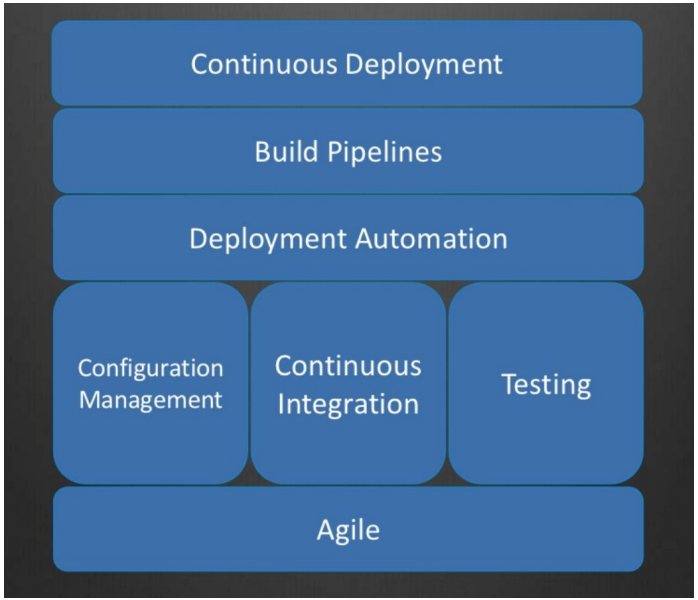
https://continuousdelivery.com/



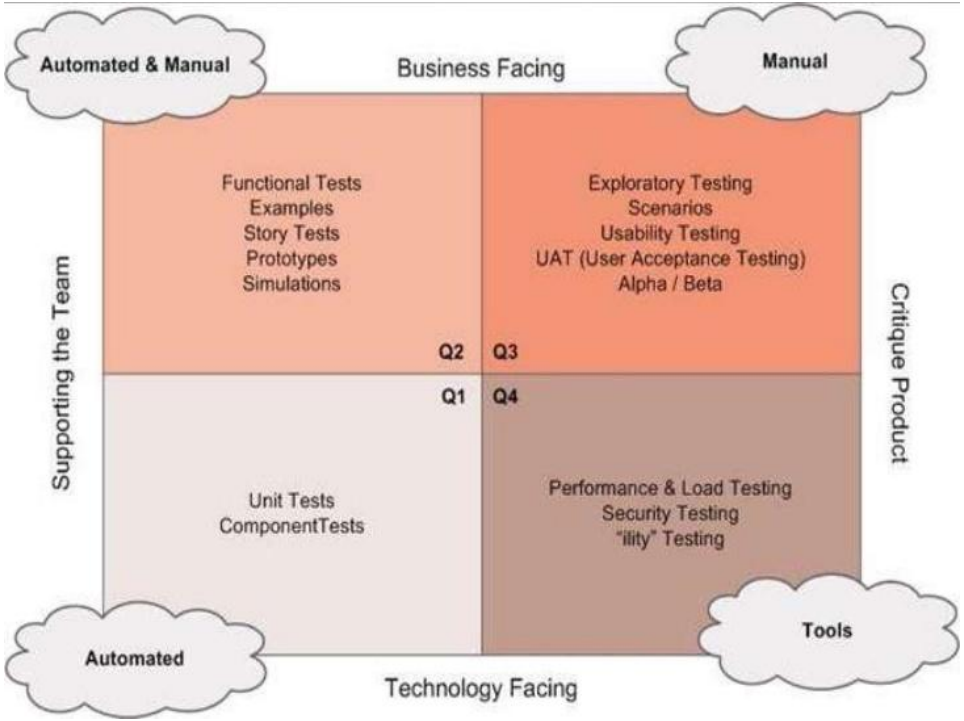
Continuous Delivery Benefits

- Ensures that every change to the system is releasable
- Lowers risk of each release - makes releases “boring”
- Delivers value more frequently
- Get fast feedback on what users care about

Continuous Delivery Practices



Continuous Delivery Practices

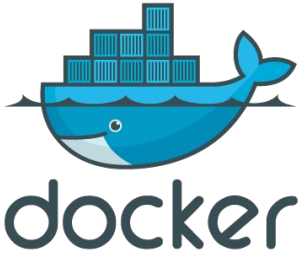


CI/CD Tools & System

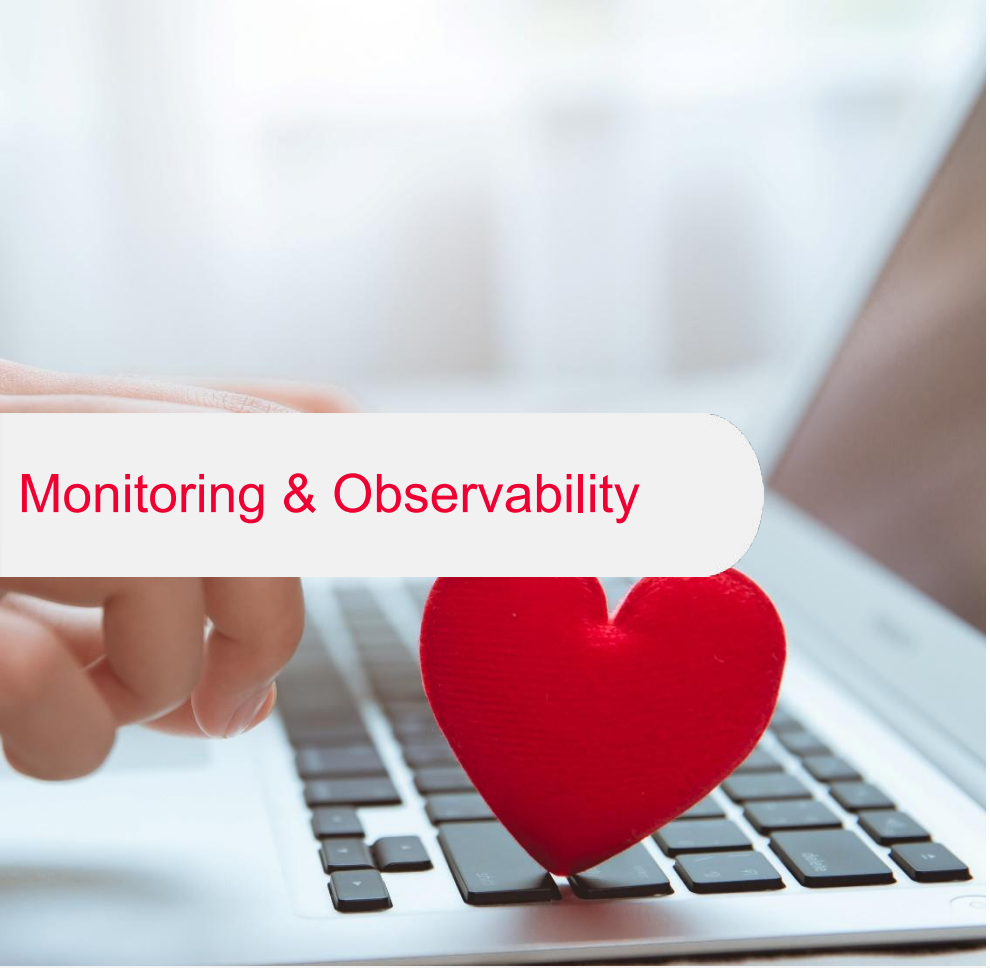
- Automation testing tools
  - Unit Test
  - Component Test
  - Performance Test
  - Acceptance Test
  - UI Test
  - ...
- Automation code analysis tool

CI/CD Tools & System

- Source Version Control System
- Packaging Tool
- Application Repository
- Integration System
- Orchestration System



Monitoring & Observability

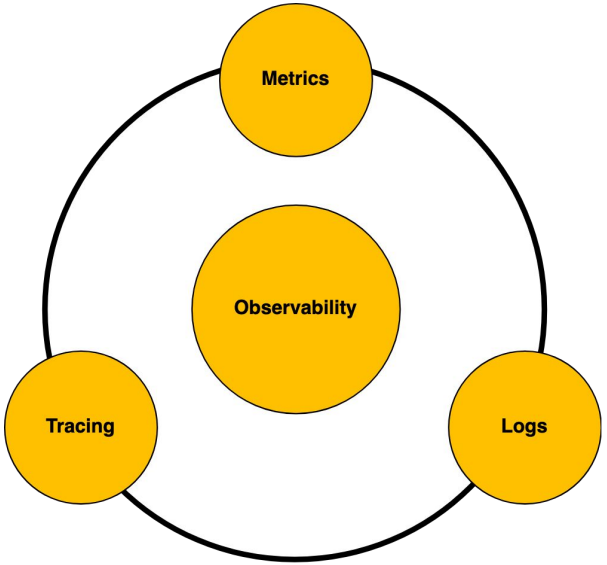


What is Observability?

- There is no “real” definition for Observability
- The ability to understand what is happening inside of a system from the knowledge of its external outputs - Dynatrace
- “We use it to describe the tools, data sources, and methods for understanding (observing!) how a technology is operating” - Brendan Gregg

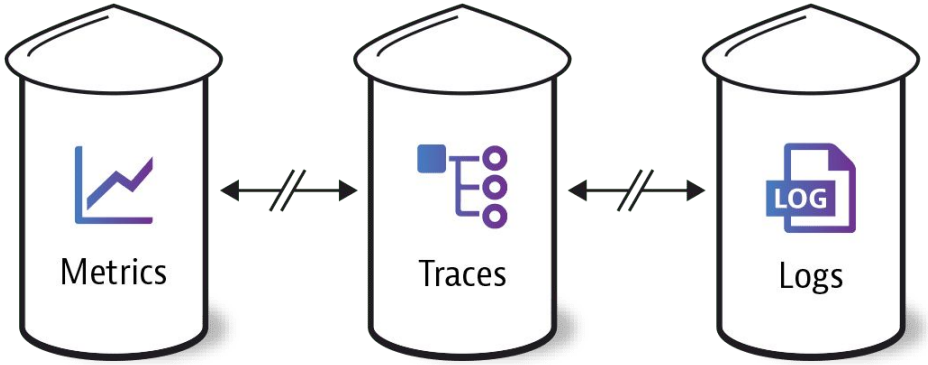
What is Observability?

- for distributed systems, such as microservices, serverless, service meshes, etc., these outputs are telemetry data



Three pillars of Telemetry

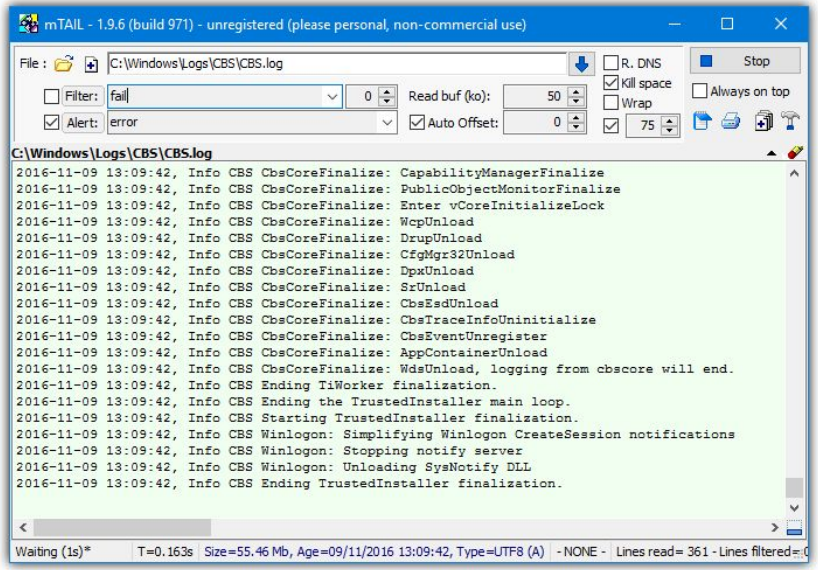
- Telemetry data is divided into three major verticals – metrics, logs, and distributed traces



Source: Dynatrace

Logs

- an event log is an immutable, timestamped record of discrete events that happened over time
- easiest to generate (printf, echo)
- no external tools required
- human readable
- UNIX-style
- can affect application performance as a whole
- system scoped

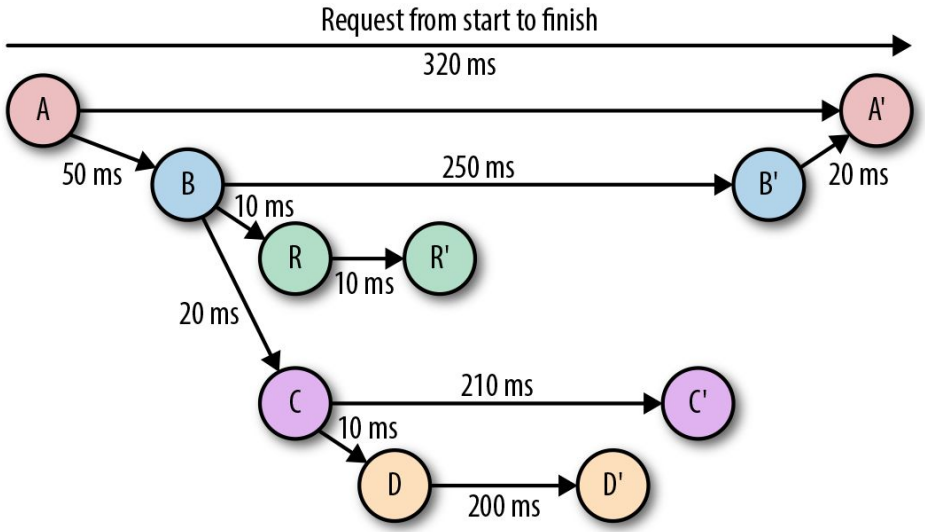


Metrics

- metrics are a numeric representation of data measured over intervals of time
- minimal store size
- low performance impact
- easier to query
- overheads are predictable
- suitable for triggering alerts
- system scoped

Tracing

- a trace is a representation of a series of causally related distributed events that encode the end-to-end request flow through a distributed system
- a representation of logs
- an understanding of the entire request lifecycle
- inter service dependency analysis
- distributed profiling
- debugging steady-state problems
- everything requires additional instrumentation



Why Observability?

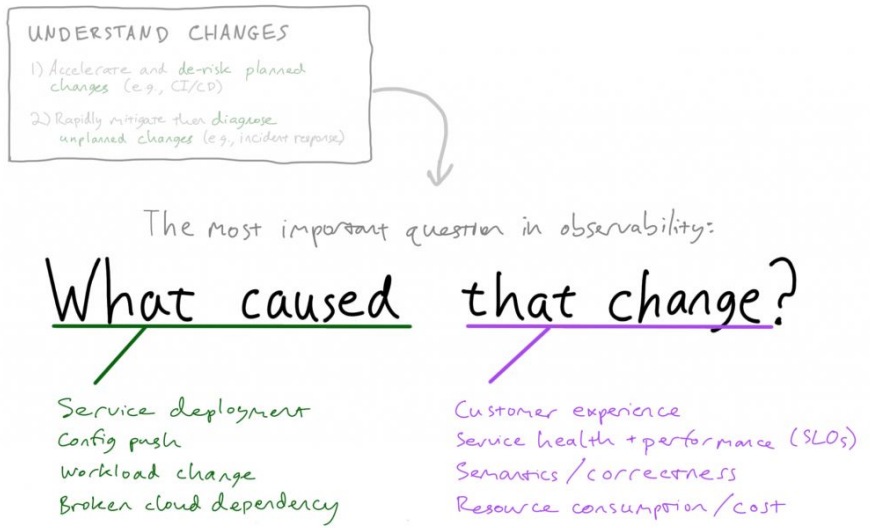
- there are two overarching problems worth solving with Observability for modern, distributed software applications:
  - Understanding Health
  - Understanding Change

Understand Health

- Connecting the well-being of a subsystem back to the goals of the overarching application and business via thoughtful monitoring

Understand Change

- Accelerating planned changes while mitigating the effects of unplanned changes

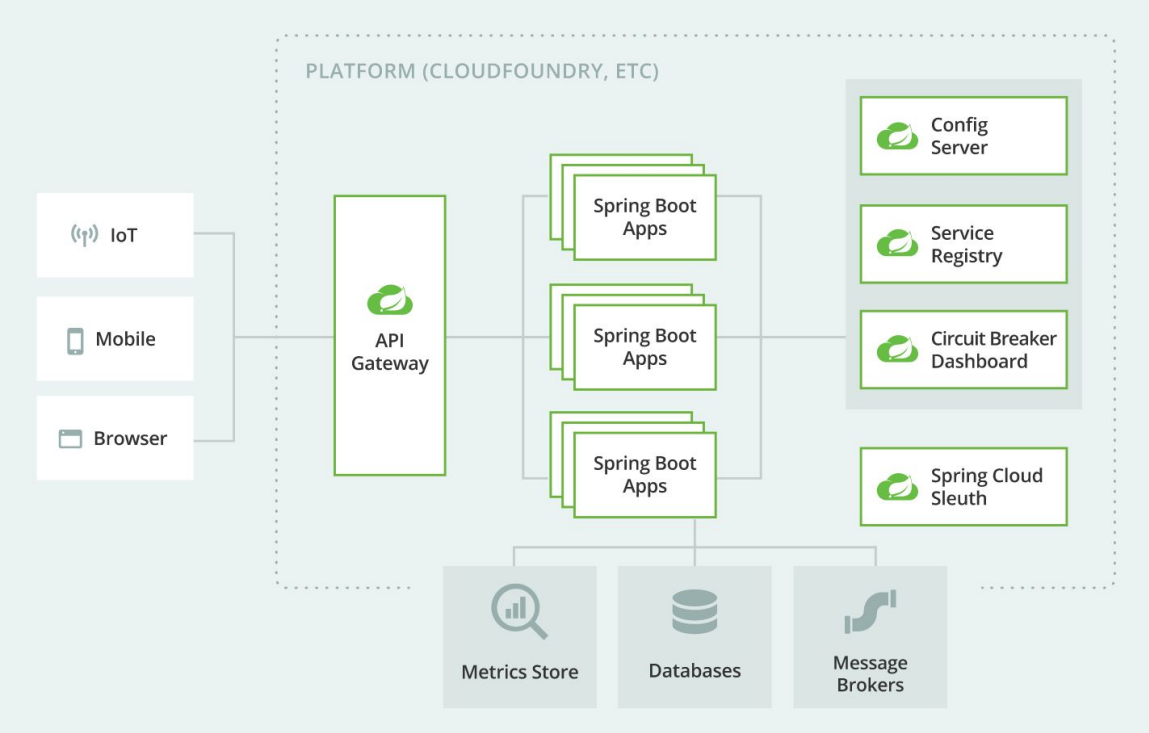


Source: [thenewstack]

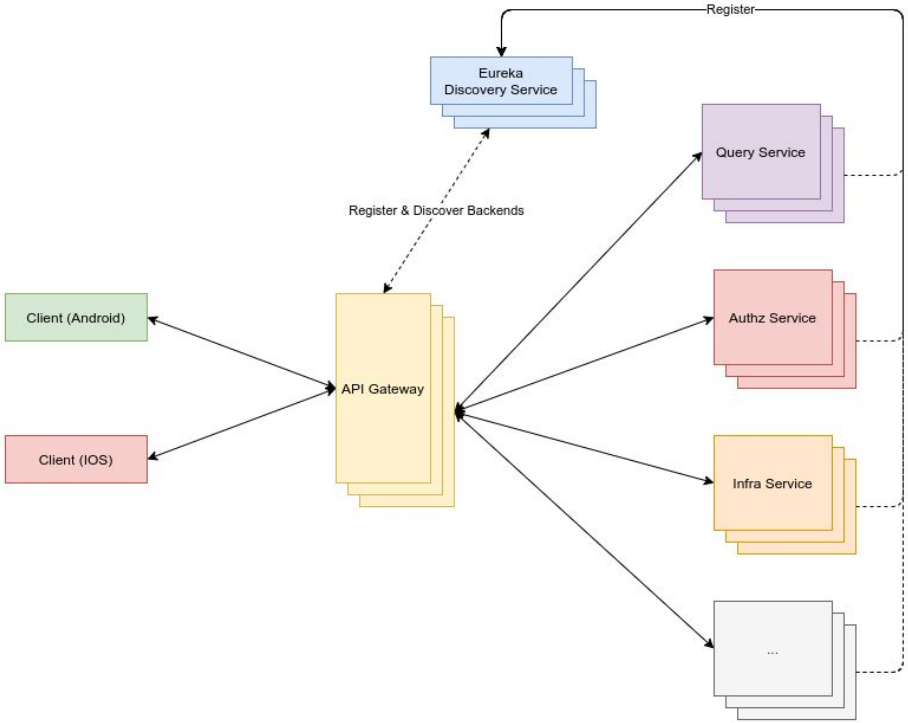
Case Study: Viettel Vsmart

- Microservices Architecture
- Cloud-Native Application
- CI/CD
- Monitoring & Observability

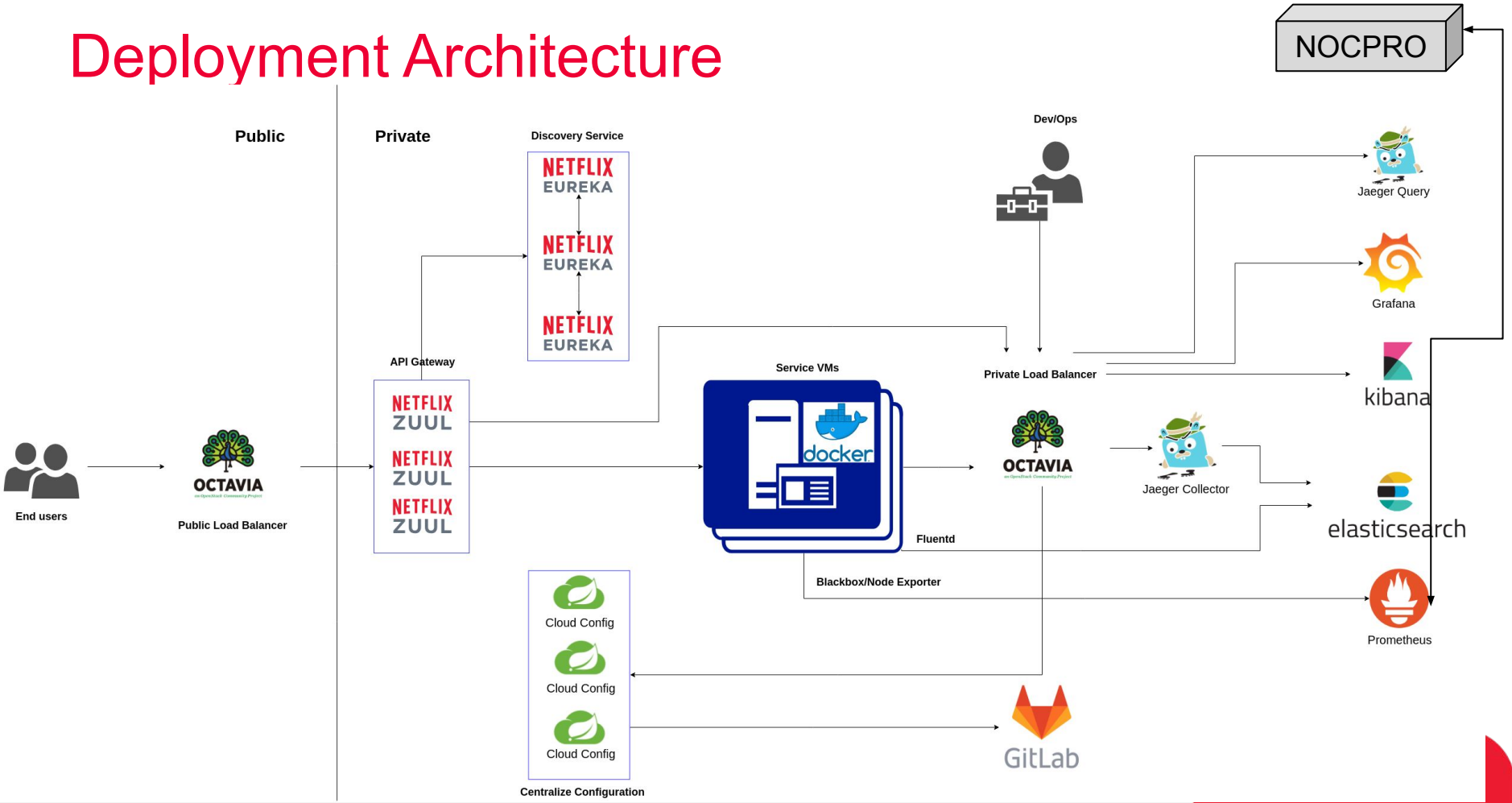
Spring Cloud Platform



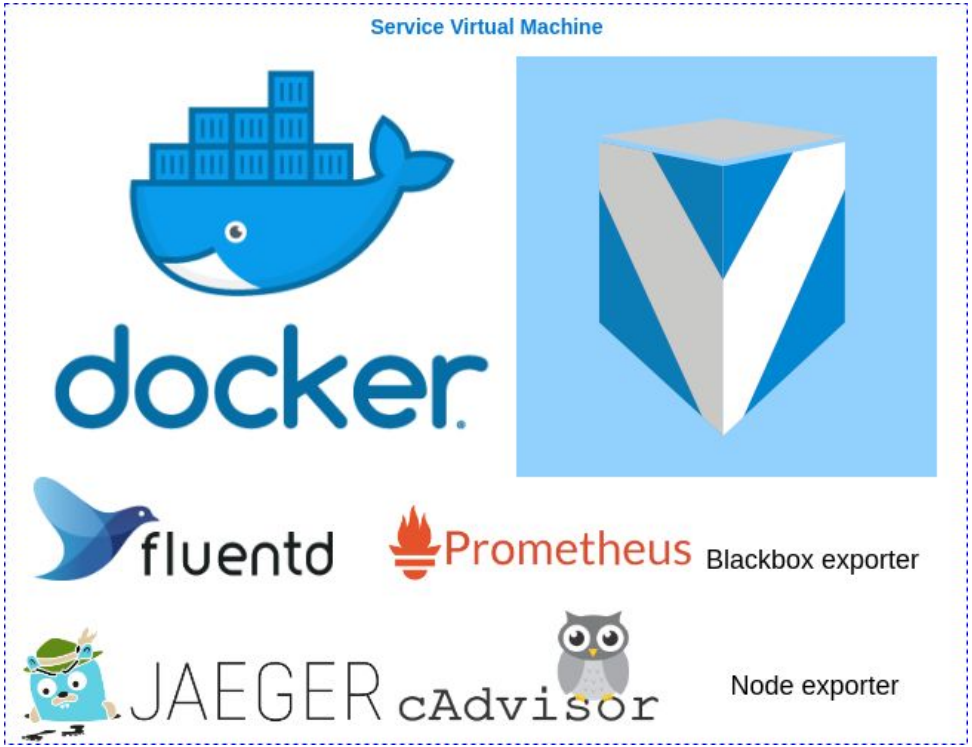
Microservices Architecture



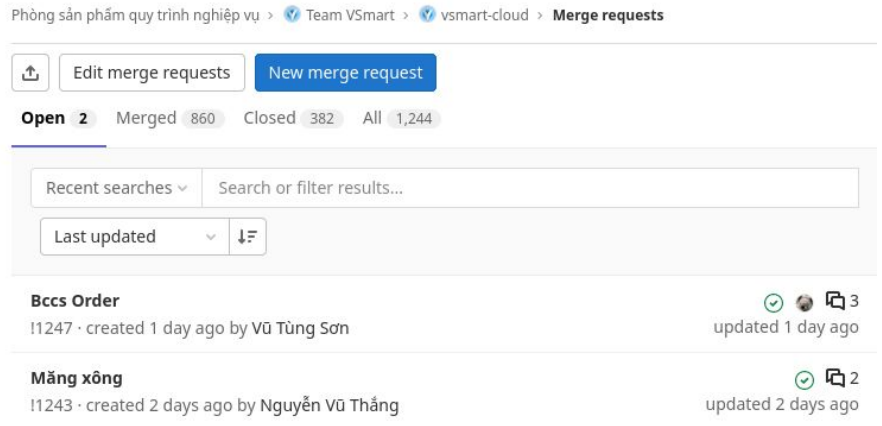
Deployment Architecture

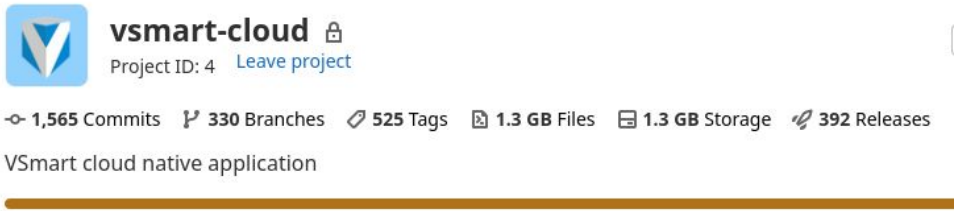


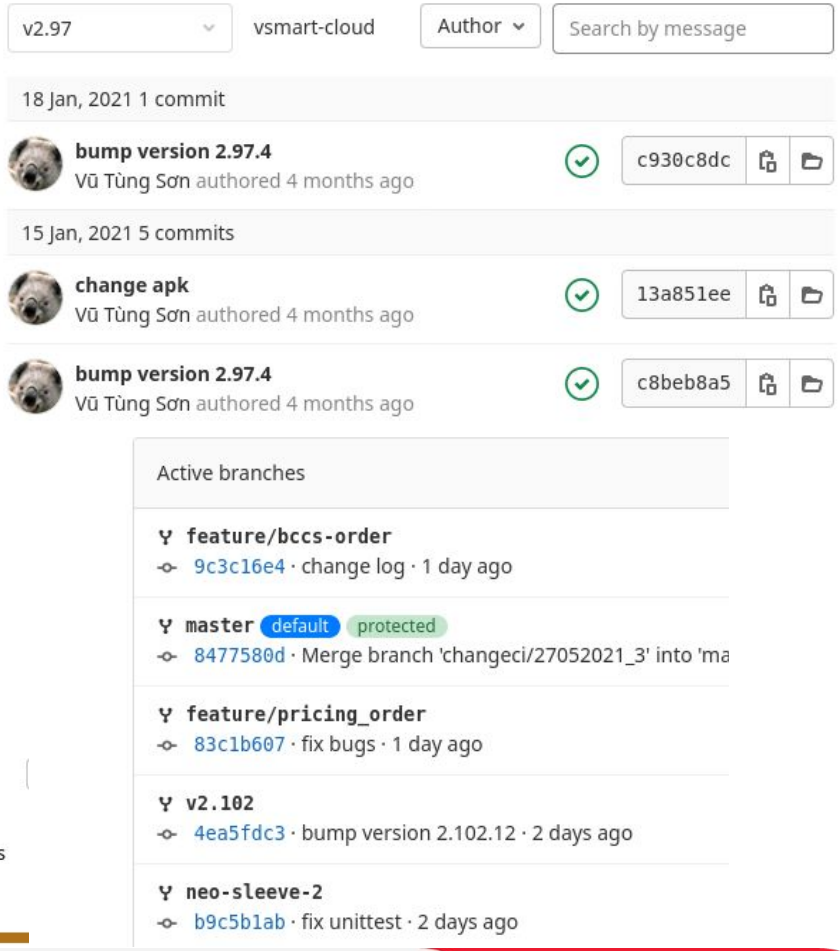
Service Instance Architecture



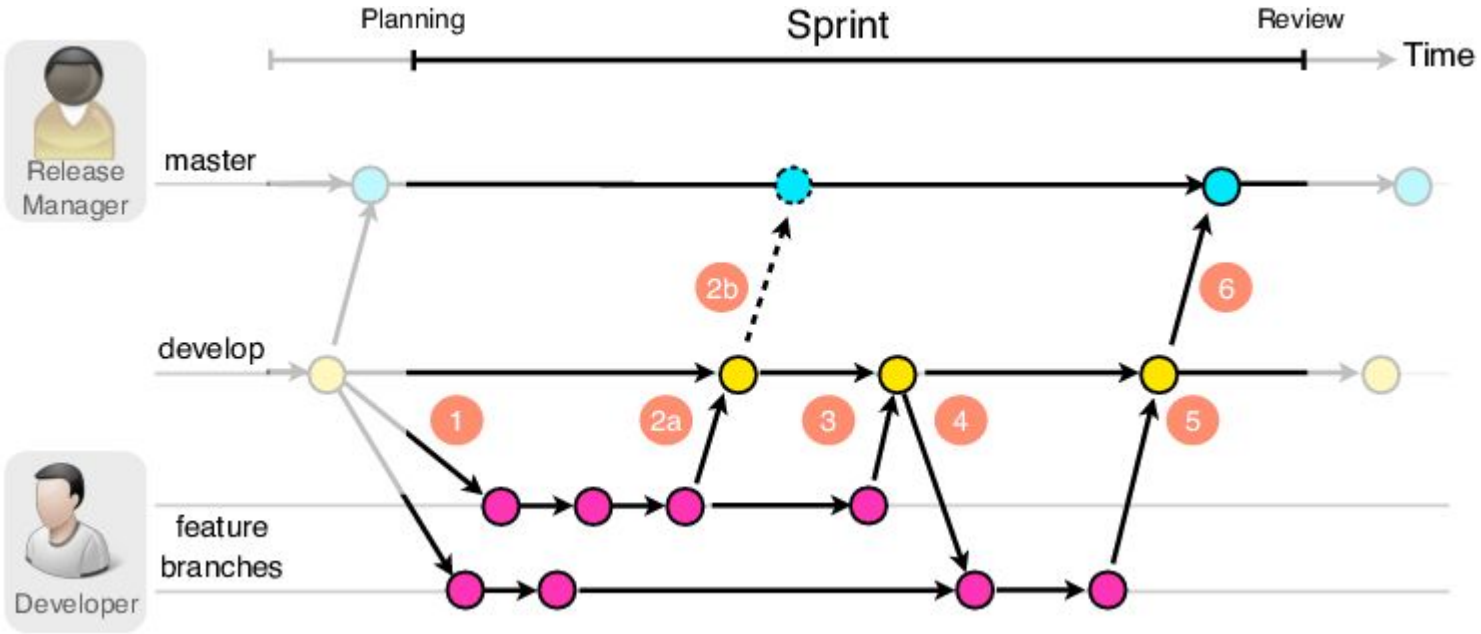
Source Code Management







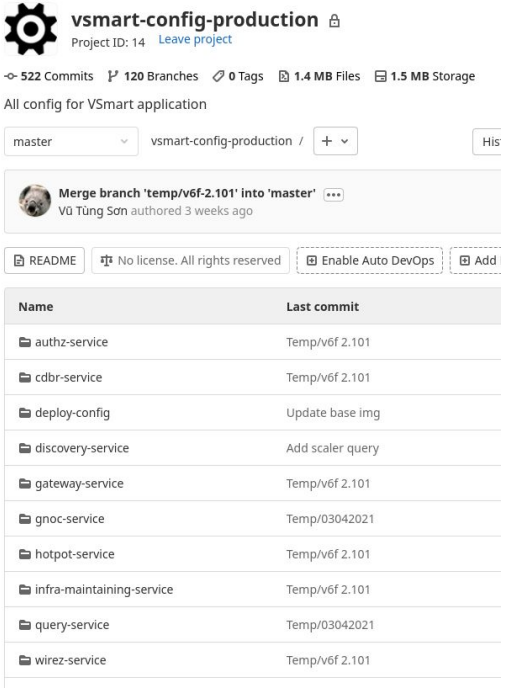
Release Model



Dependencies Management

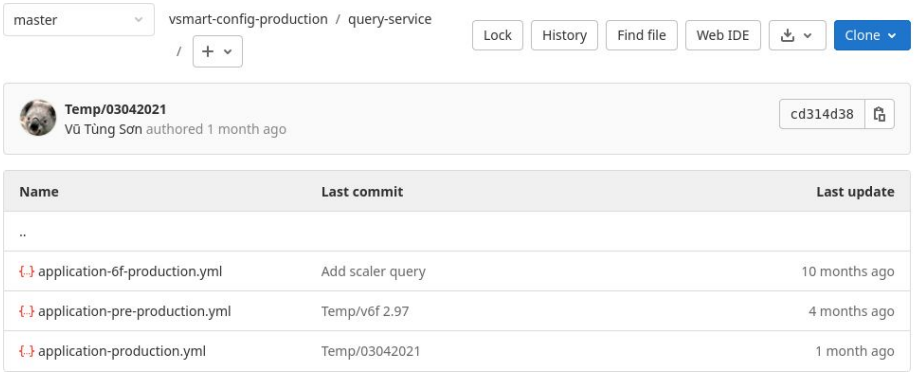
```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-web</artifactId>
</dependency>

<dependency>
  <groupId>org.springframework.cloud</groupId>
  <artifactId>spring-cloud-starter-config</artifactId>
</dependency>

<dependency>
  <groupId>org.springframework.cloud</groupId>
  <artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>
</dependency>

<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-actuator</artifactId>
</dependency>

<dependency>
  <groupId>io.micrometer</groupId>
  <artifactId>micrometer-registry-prometheus</artifactId>
</dependency>
```

Config Management

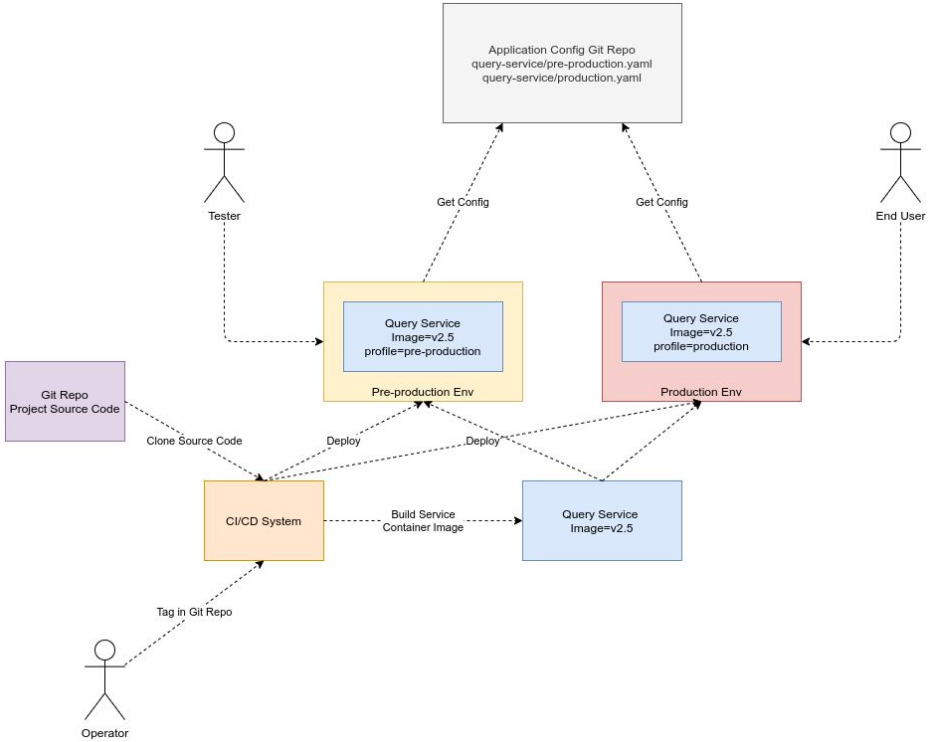



```
@Value("${application.gnoc-wfm-wo-ws-request-timeout}")
private int requestTimeout;

@Value("${application.gnoc-wfm-wo-ws-connect-timeout}")
private int connectTimeout;

@Value("${application.gnoc-wfm-wo-ws-url}")
private String urlStr;

private VSmartWS port;
```

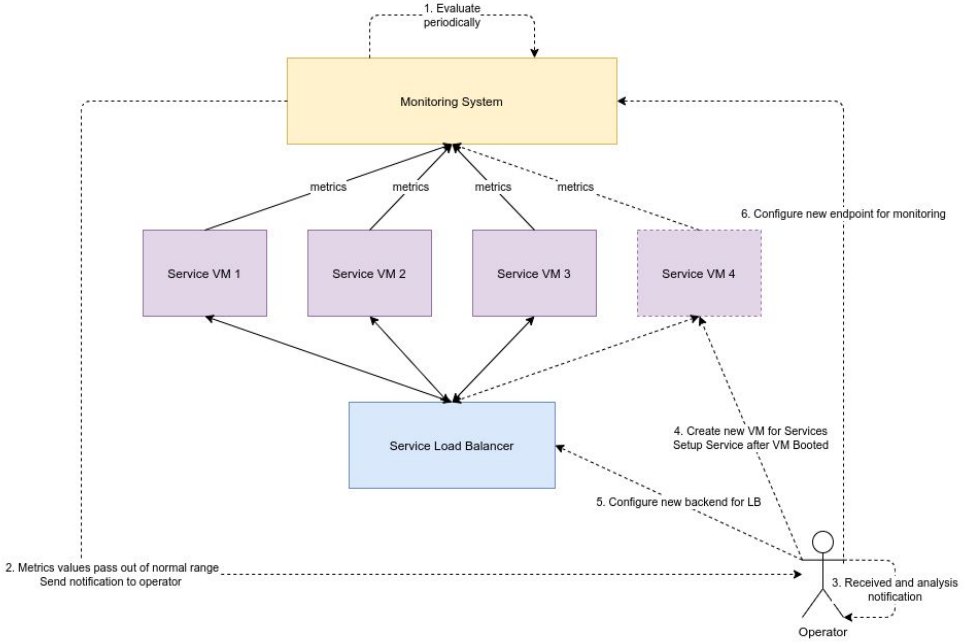
```
gnoc-wfm-wo-ws-connect-timeout: 15000
gnoc-wfm-wo-ws-request-timeout: 20000
gnoc-wfm-wo-ws-service-name: SERVICE_NAME
gnoc-wfm-wo-ws-url: http://ANOTHER_SERVICE:9992/WFMServer/DATA
```



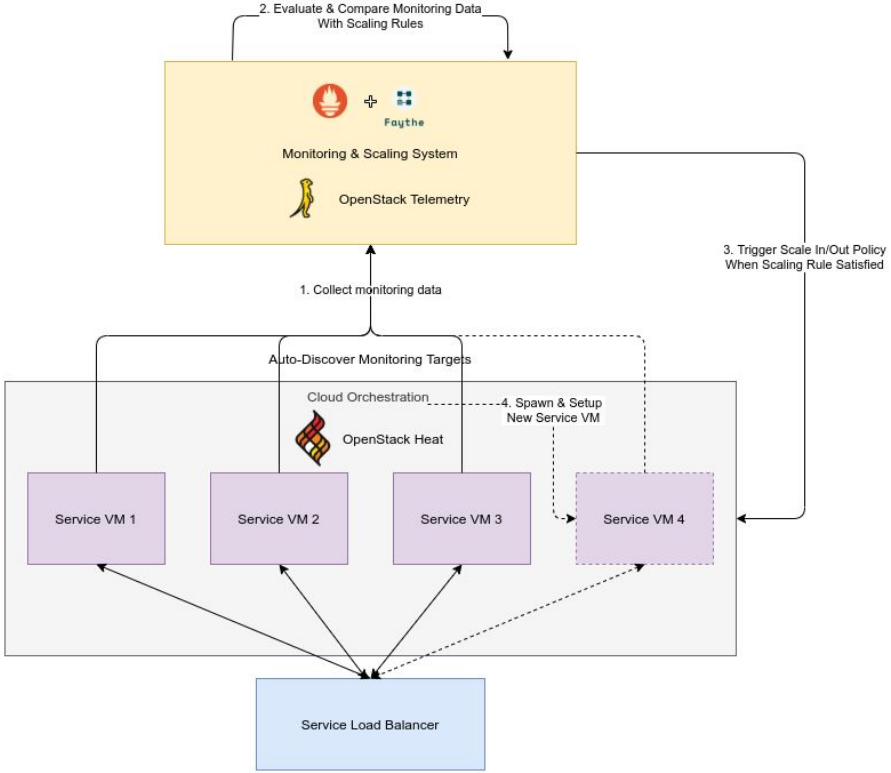
Build One - Run Everywhere



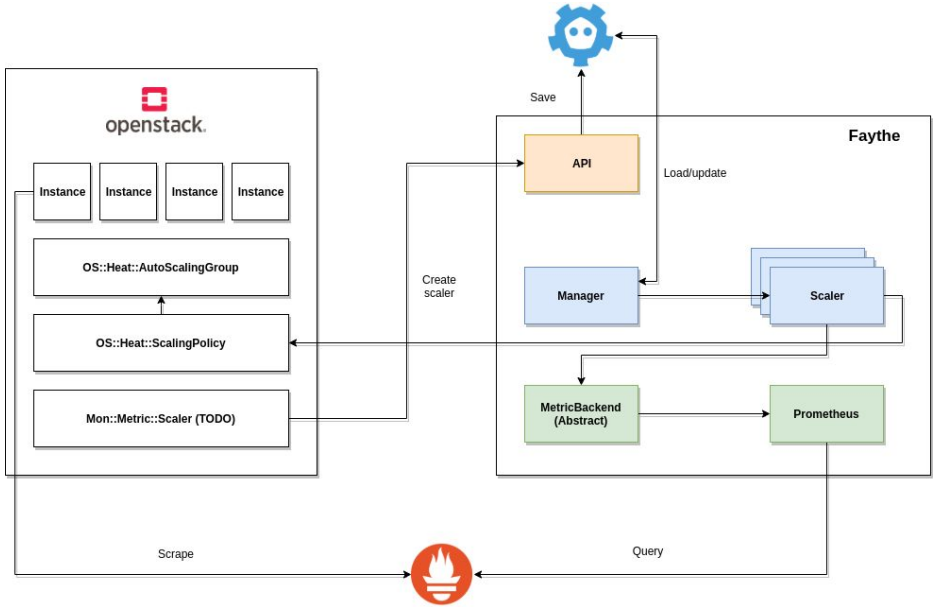
Auto Scaling In Cloud



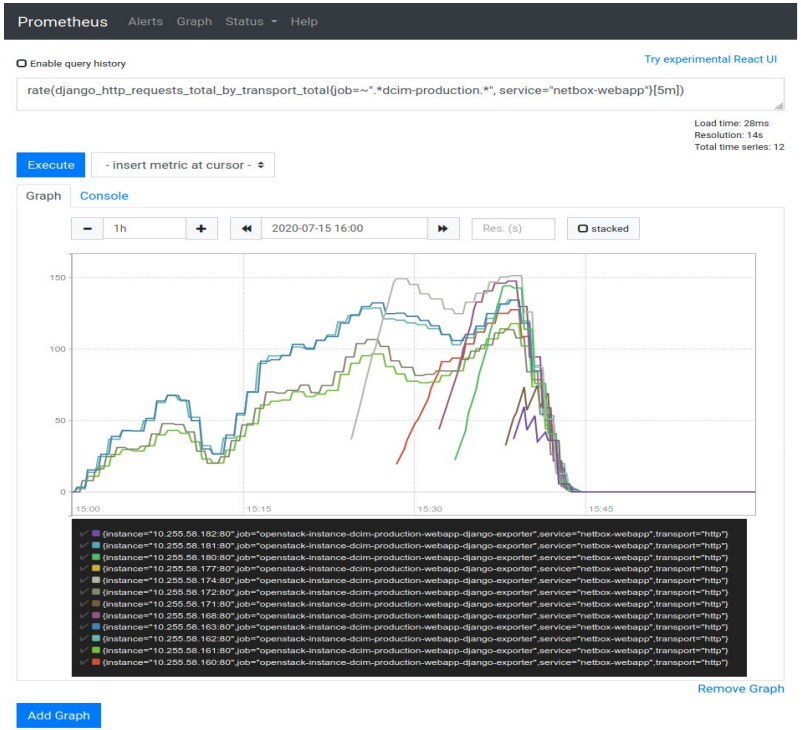
Auto Scaling In Cloud

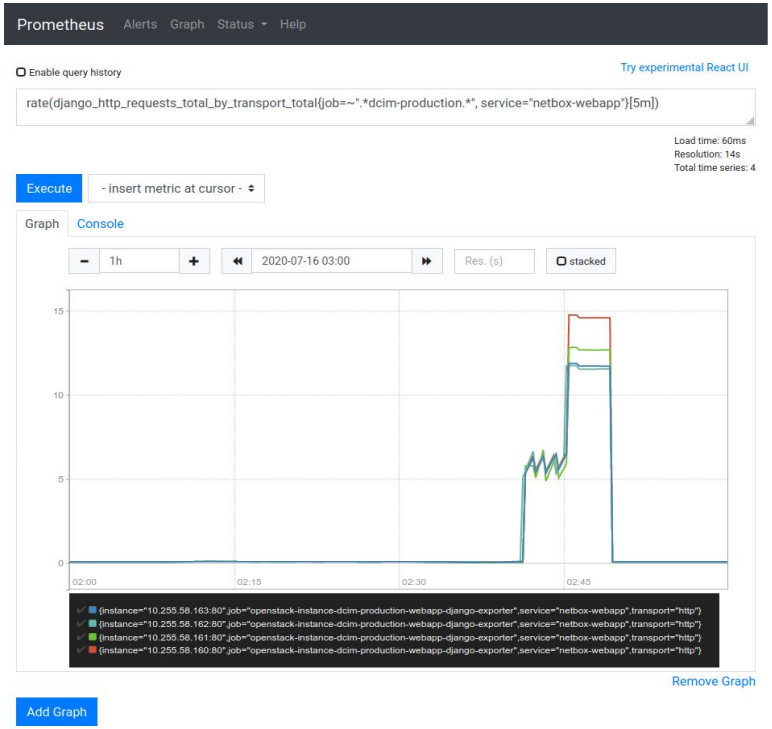


Auto Scaling In Cloud

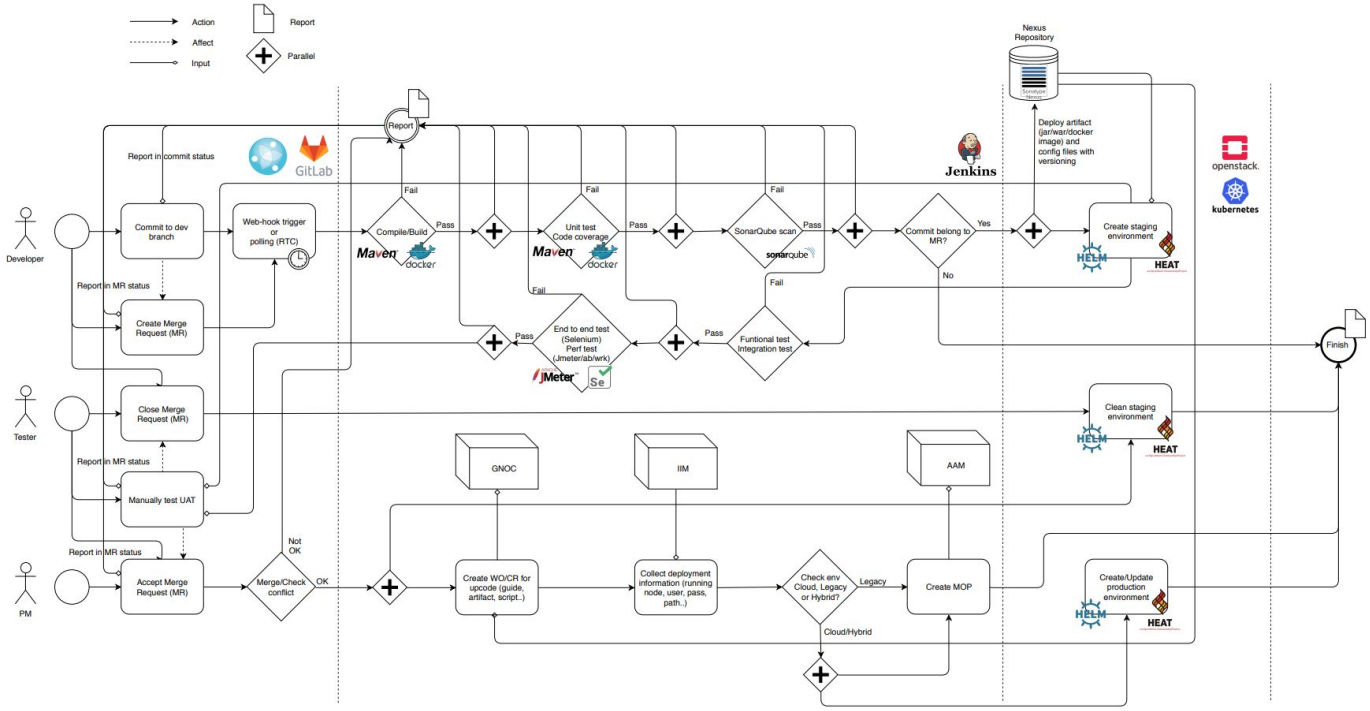


Auto Scaling In Cloud

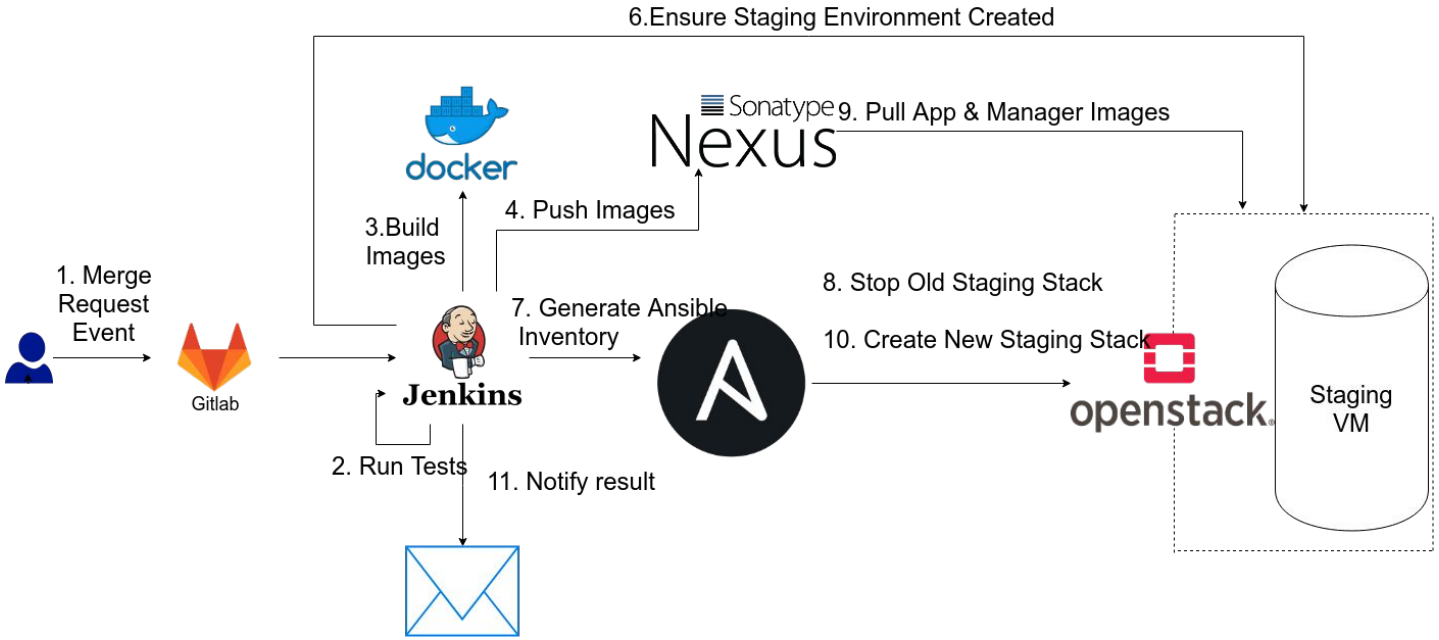




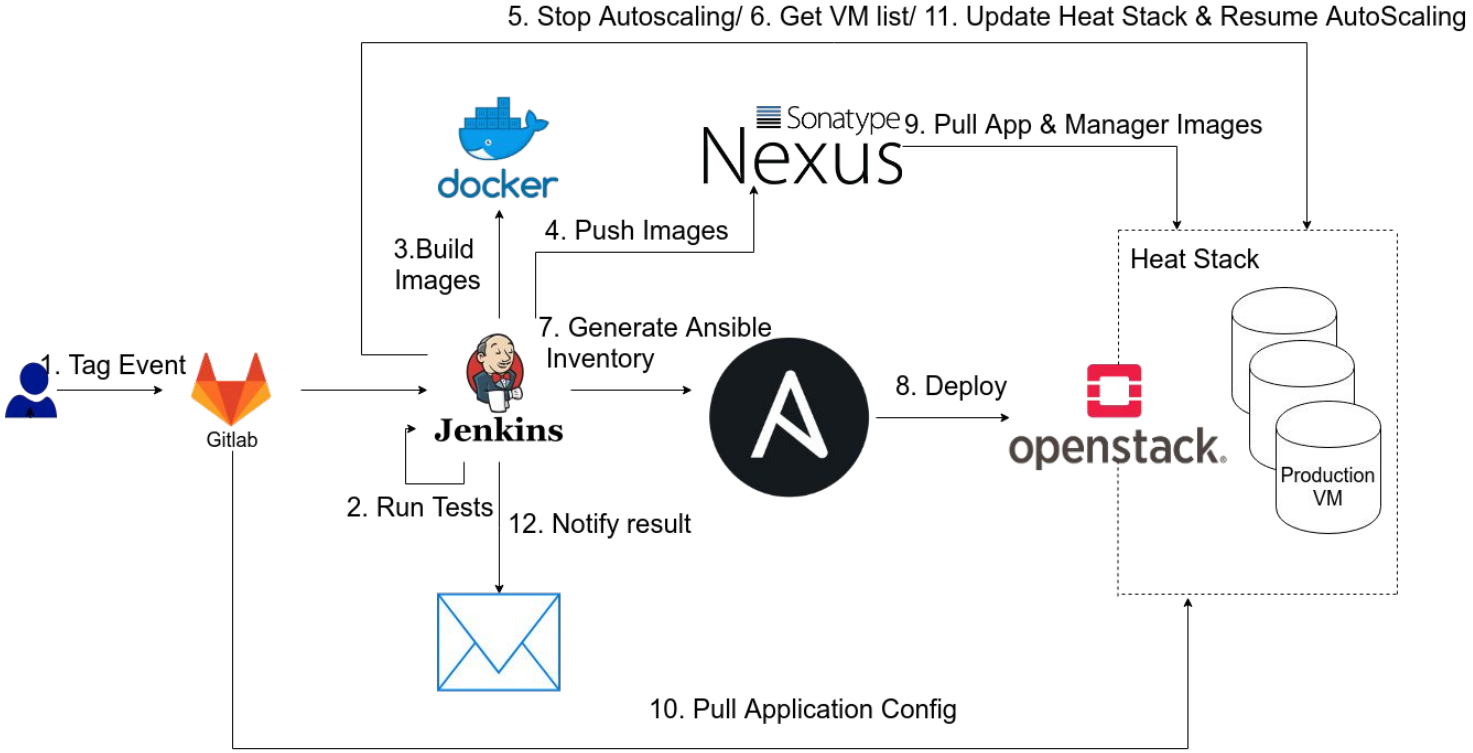
CI/CD Automation Pipeline



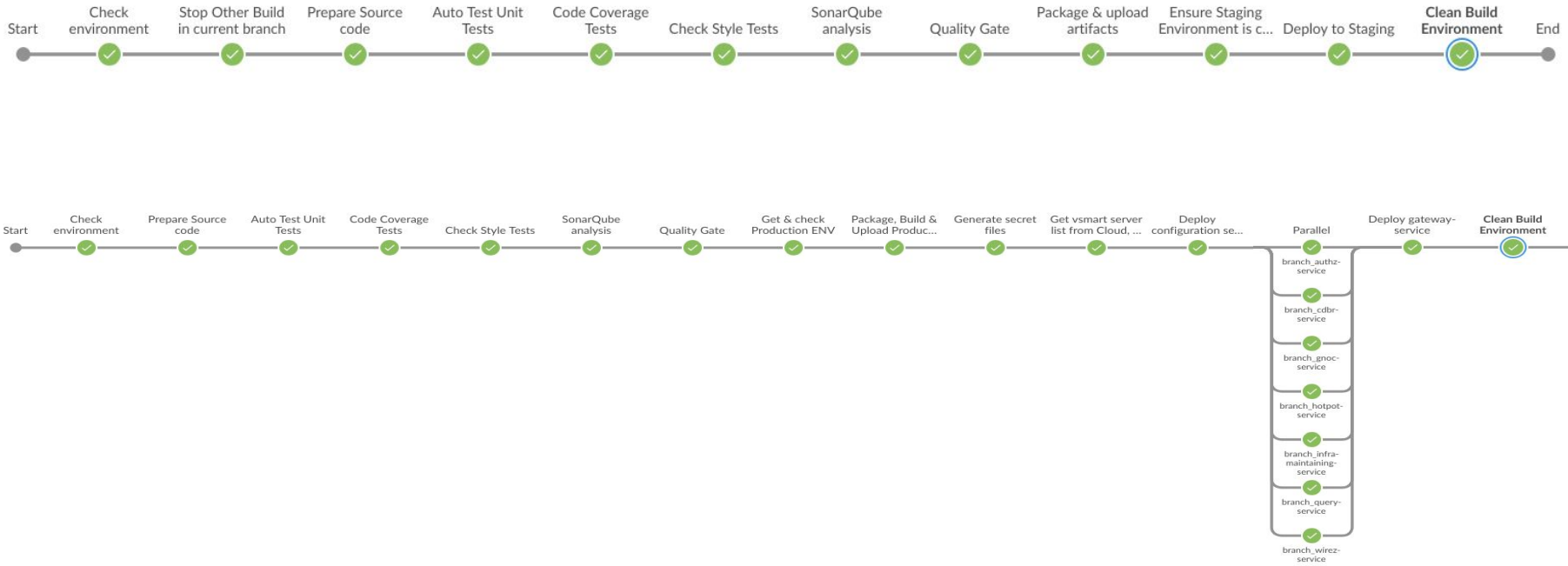
CI/CD Automation Pipeline



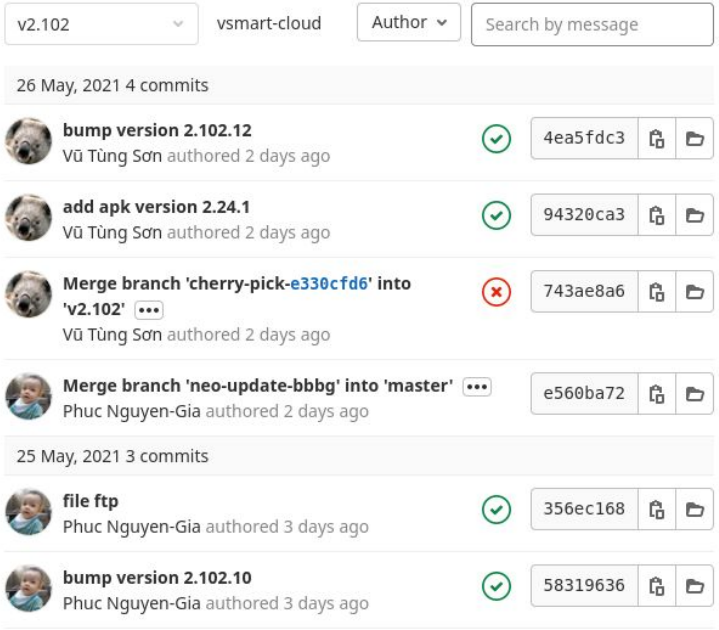
CI/CD Automation Pipeline

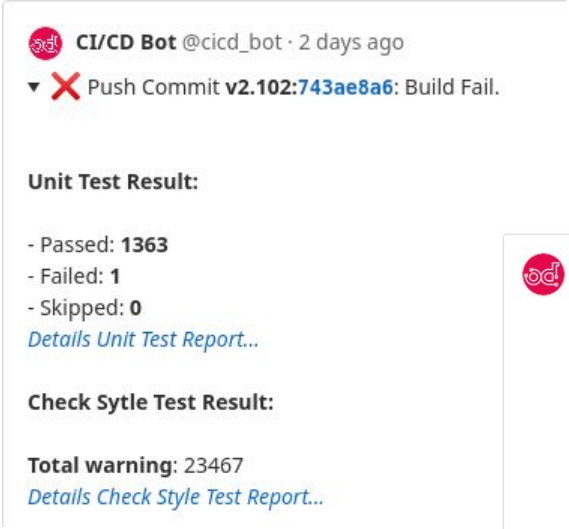


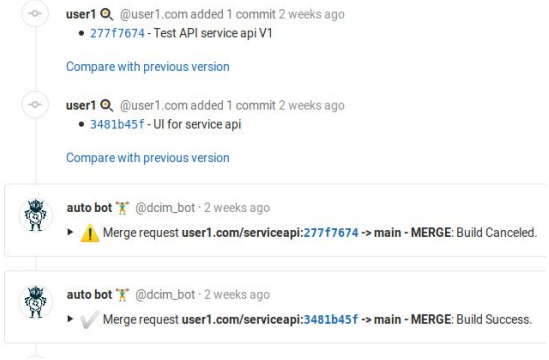
CI/CD Pipeline

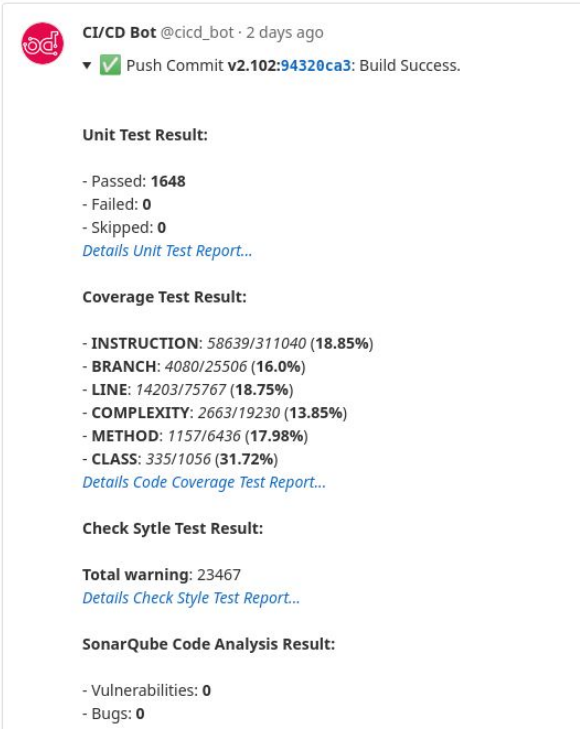


CI/CD Pipeline

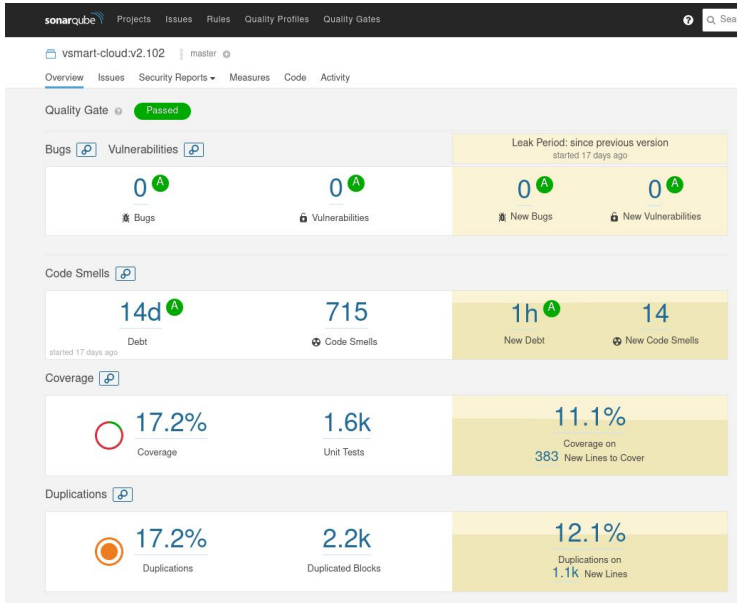








CI/CD Pipeline



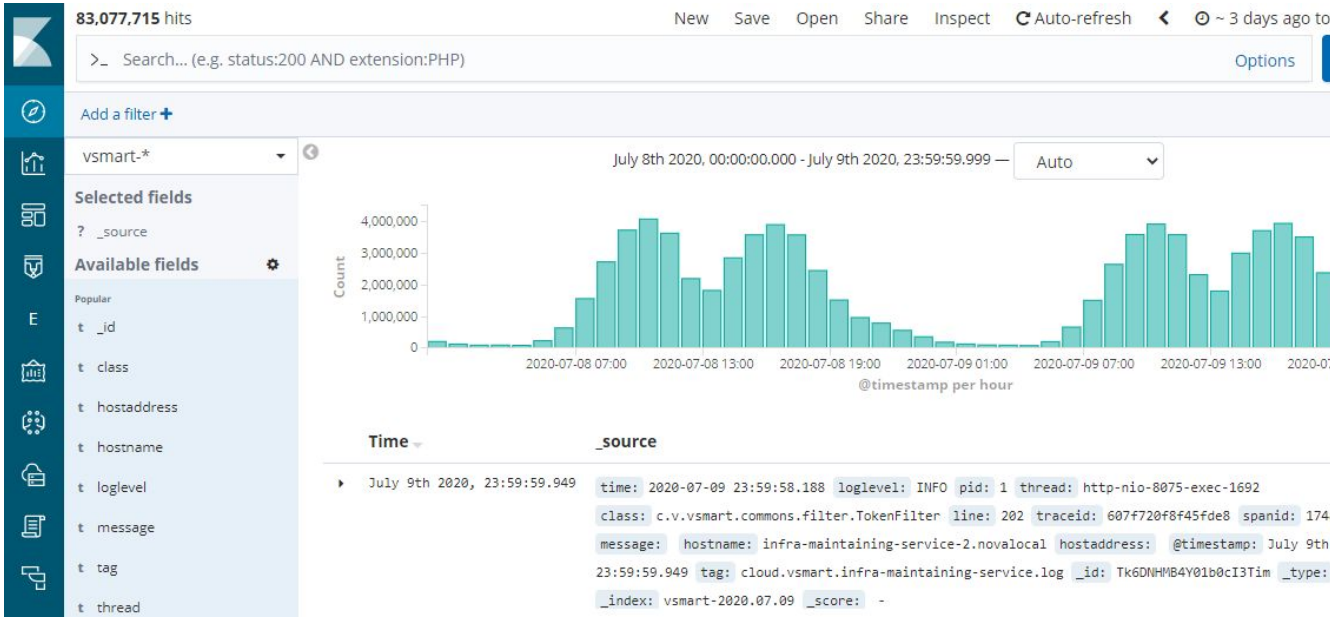
Back to #9104 Code-Coverage-Report

```
    if (Constant.TASK.INFRA_TYPE_GPON.equals(gtmbo.getTechnology())) {
      log.info("updateFinishParentBccsSale {}", taskUpdateForm);
      updateFinishParentBccsSale.process(taskUpdateForm, req, updateTKDVCDLogResult);
    } else {
      log.info("updateSetupChannelToBccsSale {}", taskUpdateForm);
      updateSetupChannelToBccsSale.process(taskUpdateForm, req, updateTKDVCDLogResult);
    }
  } else if (taskUpdateForm.getReqType() != null && taskUpdateForm.getReqType()
      .equals(Constant.TASK.TASK_TYPE_COMPLETE_CHANNEL_DATA)) {
    log.info("updateCompleteChannelToBccsSale {}", taskUpdateForm);
    updateCompleteChannelToBccsSale.process(taskUpdateForm, req, updateTKDVCDLogResult);
  } else {
    log.info("updateFinishParentBccsSale {}", taskUpdateForm);
    updateFinishParentBccsSale.process(taskUpdateForm, req, updateTKDVCDLogResult);
  }
```

Monitoring


VCLOUD-vsmart - microservice > Vsmart - vCloud Monitoring
Last 24 hours
5s
job
openstack-instance-vsmart_microservice-gnoc-service-jmx
Instance
11.22.33.44:8072
JVM Memory Pools Heap
All
JVM Memory Pools Non-Heap
All
Quick Facts
Uptime
3.2 week
Start time
2020-06-18 11:55:19
Heap used
58.74%
Non-Heap used
17.58%
I/O Overview
Rate
HTTP Current: 1.12 ops
Errors
No data
Duration
HTTP - MAX Current: 20 s
Utilisation
TOMCAT - BUSY Current: 1
TOMCAT - CURRENT Current: 18
TOMCAT - MAX Current: 200
JVM Memory
JVM Heap
used Max: 1.020 GiB Current: 594 MiB
max Max: 1.113 GiB Current: 1.113 GiB
JVM Non-Heap
used Max: 223 MiB Current: 222 MiB
committed Max: 280 MiB Current: 280 MiB
JVM Total
used Max: 1.250 GiB Current: 1.203 GiB
committed Max: 1.387 GiB Current: 1.387 GiB
JVM Native Memory
No data

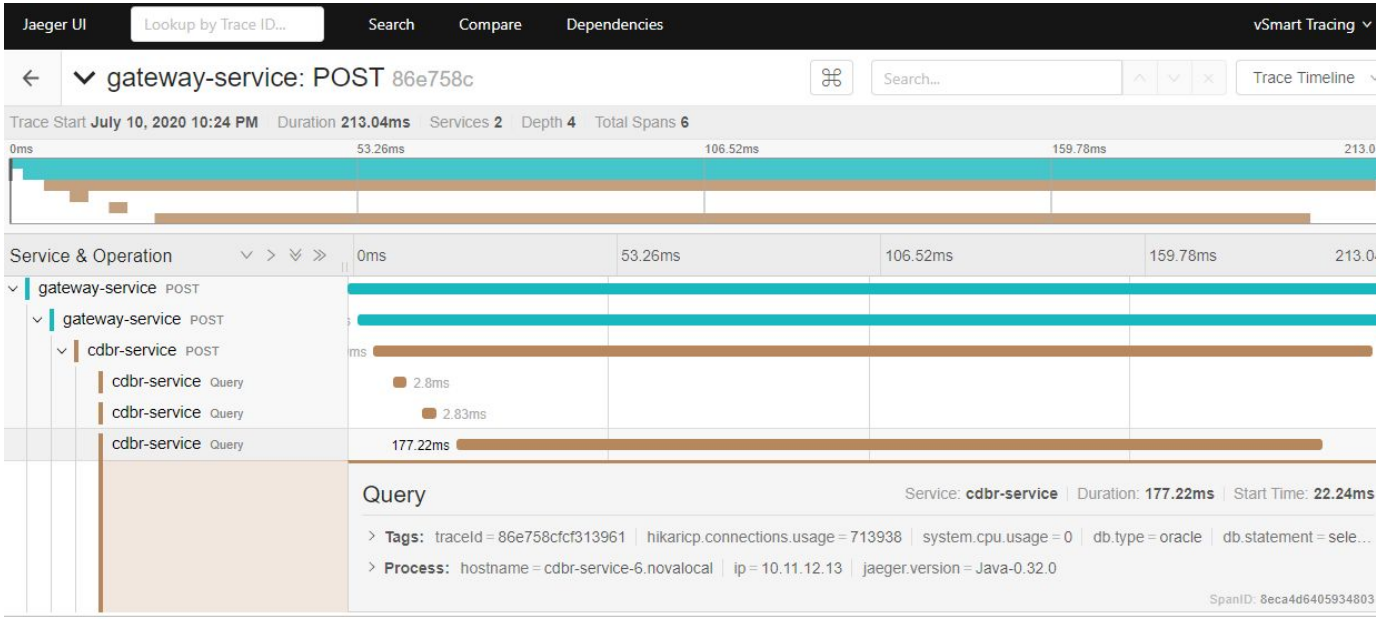
Logging


83,077,715 hits
New
Save
Open
Share
Inspect
Auto-refresh
~ 3 days ago to
Search... (e.g. status:200 AND extension:PHP)
Options
Add a filter
vsmart-*
Selected fields
? _source
Available fields
Popular
t _id
t class
t hostaddress
t hostname
t loglevel
t message
t tag
t thread
July 8th 2020, 00:00:00.000 - July 9th 2020, 23:59:59.999 —
Auto
Count
4,000,000
3,000,000
2,000,000
1,000,000
0
2020-07-08 07:00
2020-07-08 13:00
2020-07-08 19:00
2020-07-09 01:00
2020-07-09 07:00
2020-07-09 13:00
@timestamp per hour
Time
_source
July 9th 2020, 23:59:59.949
time: 2020-07-09 23:59:58.188 loglevel: INFO pid: 1 thread: http-nio-8075-exec-1692
class: c.v.vsmart.commons.filter.TokenFilter line: 202 traceid: 607f720f8f45fde8 spanid: 174
message: hostname: infra-maintaining-service-2.novalocal hostaddress: @timestamp: July 9th
23:59:59.949 tag: cloud.vsmart.infra-maintaining-service.log _id: Tk6DNHMB4Y01b0cI3Tim _type:
_index: vsmart-2020.07.09 _score: -

Logging

Time	message	hostname	traceid
May 24th 2021, 15:38:57.316	user 'tanla4' use android with user-agent: Dalvik/2.1.0 (Linux; U; Android 11; SM-A515F Build/RP1A.200720.012)	gateway-service-3.novalocal	fe5b3ab137b71627
May 24th 2021, 15:38:57.321	pathInfo: /gponController/getPonLinkFromSplitter	cdbr-service-1.novalocal	fe5b3ab137b71627
May 24th 2021, 15:38:57.322	Decrypt token: 80210#tanla4#24-05-2021 08:29:01	cdbr-service-1.novalocal	fe5b3ab137b71627
May 24th 2021, 15:38:57.322	dateInToken=24-05-2021[nowDate]24-05-2021	cdbr-service-1.novalocal	fe5b3ab137b71627
May 24th 2021, 15:38:57.324	10.240.203.153_tanla4_80210_null_202105241538558720_226 ||>> /gponController/getPonLinkFromSplitter:{splitterCode:LAN0257-GN12-SN11-SP01;oltType:;token:c624ca58-8a94-4d6c-b859-6f612ef4b643;}	cdbr-service-1.novalocal	fe5b3ab137b71627
May 24th 2021, 15:38:57.333	tanla4 Route mapping: '/gponController/getPonLinkFromSplitter' ---> 'gpon-Controller-get-pon-link-from-splitter' on 'cdbr-service'	gateway-service-3.novalocal	fe5b3ab137b71627
May 24th 2021, 15:38:57.355	10.240.203.153_tanla4_80210_null_202105241538558720_226 ||<< 200 OK	cdbr-service-1.novalocal	fe5b3ab137b71627
May 24th 2021, 15:38:57.357	10.240.203.153_tanla4_80210_null_202105241538558720_226 ||<< {"ponLinkFromSplitter":{"stationId":"32460","stationCode":"LAN0257","device":"13691319","deviceCode":"LAN0257OLT01","portCode":"1-1-6-4","ampDeviceCode":"LAN0257APA01","ampDeviceId":13691293,"nodeBranchCode":"LAN0257-GN12-SN11","nodeBranchId":15934620,"nodeSubCode":"LAN0257-GN12-SN11","nodeSubId":15934620,"splitterCode":"LAN0257-GN12-SN11-SP01","splitterId":15935986,"portAmpCode":"0/0/22(OUT)","isCheckCompalintTask":false}}	cdbr-service-1.novalocal	fe5b3ab137b71627
May 24th 2021, 15:38:57.470	QLCTKT_CODE|/gponController/getPonLinkFromSplitter|fe5b3ab137b71627: 36c2e891c5a1c2aa|10.240.203.205|10.240.203.175:8071|{splitterCode:LAN0257-GN12-SN11-SP01;oltType:;token:c624ca58-8a94-4d6c-b859-6f612ef4b643;}|{"ponLinkFromSplitter":{"stationId":"32460","stationCode":"LAN0257","device":"13691319","deviceCode":"LAN0257OLT01","portCode":"1-1-6-4","ampDeviceCode":"LAN0257APA01","ampDeviceId":13691293,"nodeBranchCode":"LAN0257-GN12-SN11","nodeBranchId":15934620,"nodeSubCode":"LAN0257-GN12-SN11","nodeSubId":15934620,"splitterCode":"LAN0257-GN12-SN11-SP01","splitterId":15935986,"portAmpCode":"0/0/22(OUT)","isCheckCompalintTask":false}}|24/05/2021 03:38:57|24/05/2021 03:38:57|34|200|OK|1|/gponController/getPonLinkFromSplitter|tanla4||01|87|427	cdbr-service-1.novalocal	fe5b3ab137b71627

Tracing


Jaeger UI
Lookup by Trace ID...
Search
Compare
Dependencies
vSmart Tracing
gateway-service: POST 86e758c
Search...
Trace Timeline
Trace Start July 10, 2020 10:24 PM
Duration 213.04ms
Services 2
Depth 4
Total Spans 6
0ms
53.26ms
106.52ms
159.78ms
213.0
Service & Operation
gateway-service POST
gateway-service POST
cdbr-service POST
cdbr-service Query
cdbr-service Query
cdbr-service Query
2.8ms
2.83ms
177.22ms
Query
Service: cdbr-service
Duration: 177.22ms
Start Time: 22.24ms
Tags: traceId = 86e758cfcf313961
hikaricp.connections.usage = 713938
system.cpu.usage = 0
db.type = oracle
db.statement = sele...
Process: hostname = cdbr-service-6.novalocal
ip = 10.11.12.13
jaeger.version = Java-0.32.0
SpanID: 8eca4d6405934803

References

- https://www.nginx.com/blog/introduction-to-microservices/
- https://12factor.net/
- https://martinfowler.com/articles/microservices.html
- Ebook: Building Microservices - Sam Newman

Exercise

- Practice Jenkins CI example: https://www.jenkins.io/doc/tutorials/build-a-java-app-with-maven/
- Write a simple Web Server in a language (Go, Java, Python, Ruby...)
- Write some unit tests for above Web Server
- Push Code to Repository
- Write CI Pipeline in file **Jenkinsfile**, perform unit test in CI Pipeline
- Add Dockerfile and docker-compose.yml to repository
- Write CD Pipeline in file **Jenkinsfile-CD** performs:
  - Build docker image from Repository
  - Push docker image to DockerHub
- Install ansible plugins to Jenkins: https://plugins.jenkins.io/ansible/
- Update CD Pipeline, add **deploy stage** using ansible deploy web server docker-compose stack to a target host (localhost or VM)
- Submit Web Server Source Code, two file **Jenkinsfile** and **Jenkinsfile-CD** to **week-5** folder, include Jenkins run CI and CD Console Output logs if you have

THANK YOU

BACKUP SLIDES

Testing in Microservice

https://martinfowler.com/articles/microservice-testing/#anatomy-modules

Demo CI/CD Pipeline

https://github.com/haminhcong/simple-java-maven-app/blob/master/Jenkinsfile
http://127.0.0.1:8080/