Microservices from DevOps perspective

Main topics

- What are Microservices and why use them?
- What is DevOps and how is related?
- What are the challenges then?
- Where to start our CICD planification? What to consider?
- Where to go?

Microservices

Definition

Wikipedia: *A microservice architecture is an architectural pattern that arranges an application as a collection of loosely-coupled, fine-grained services, communicating through lightweight protocols.*

…application architecture that breaks an application into various service components compared to a traditional monolithic architecture…

Microservices solve the challenges of monolithic systems by being as modular as possible…

…development method that breaks down software into modules with specialized functions and detailed interfaces…

Microservices

Why to use them?

- **Fault Isolation and Application Resiliency**

Features and information domains are separated into different services or API's, so if one is affected the rest of the application will still working.

- **Data isolation**

Each *information or data domain* and its related logic will be owned by a single service. This carries with benefits as simplifying schemas updates, reduce or isolate risks on database updates and help for a loosely coupled communication within services.

- **Scalability**

In a Microservices architecture each component has the ability to scale independently, by its own metrics and possible dedicated rules, resulting in a significant increase in resource consumption efficiency.

- **Independent Deployment**

Smaller and retro-compatible software increments over different services allows individual deployments, reducing the risk of affecting the entire application, outages and general downtimes if any.

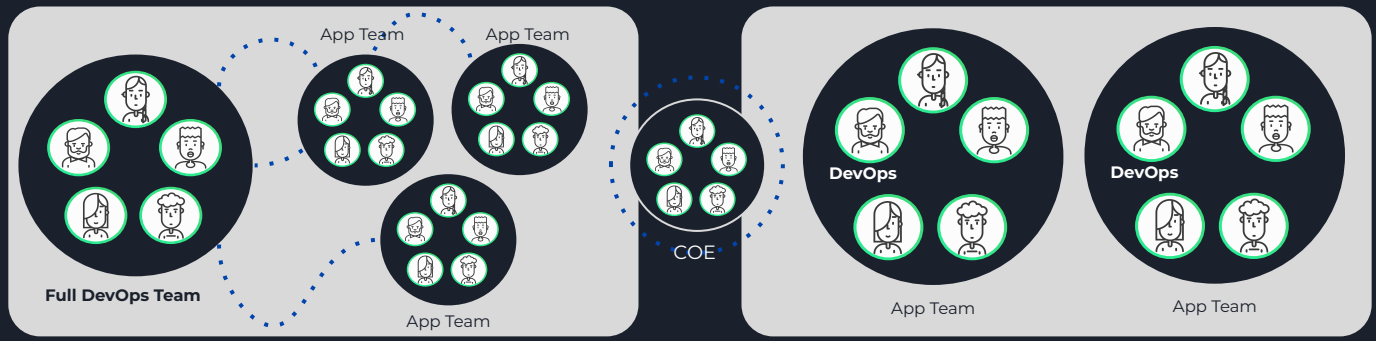
DevOps

Cultural vs Tech Industry Position

Cultural: *Is a set of practices that combines software development (Dev) and IT operations (Ops). It aims to shorten the systems development life cycle and provide continuous delivery with high software quality.*

Position: *In a more practical way is a cultural concept mistakenly converted into a software position by industry. It's a role present into an Application Development team principally in charge of CICD automation.*

Operational Model



DevOps

Microservices and DevOps

App Team

Microservices architecture based solution

- **Firewall**
- **API Management**
- **Load balancers**
- **DNS**
- **Databases**
- **Cache**
- **Messages**
- **Logs**
- **Traces**
- **Infrastructure**
- **Environments**
- **Repositories + Branches**
- **Versioning**
- **Pipelines**
- **Quality**

DevOps

Challenges with Microservices

- **Promoting code through Branches**

Mostly in multi-repository than single-repository scenarios, though either, App Teams will lead with merging code from features and hotfixes branches to mains and depending on how deployments are triggered this frequently could consumed an important amount of time.

- **High complex deployment scenarios**

Releases with multiples Microservices involved getting aligned on each environment for quality and user validation, each of those with its schemas updates, configuration changes either environment and/or application.

- **Retro-compatibility**

Microservices features into small code iterations with a retro-compatibility development strategy in order to allow independent deployments not affecting not current consumers.

- **Quality assurance**

Validate features or software increment iterations, look for application regressions over multiples environments always aiming for having symmetrical services and schemas scenarios.

- **Databases updates**

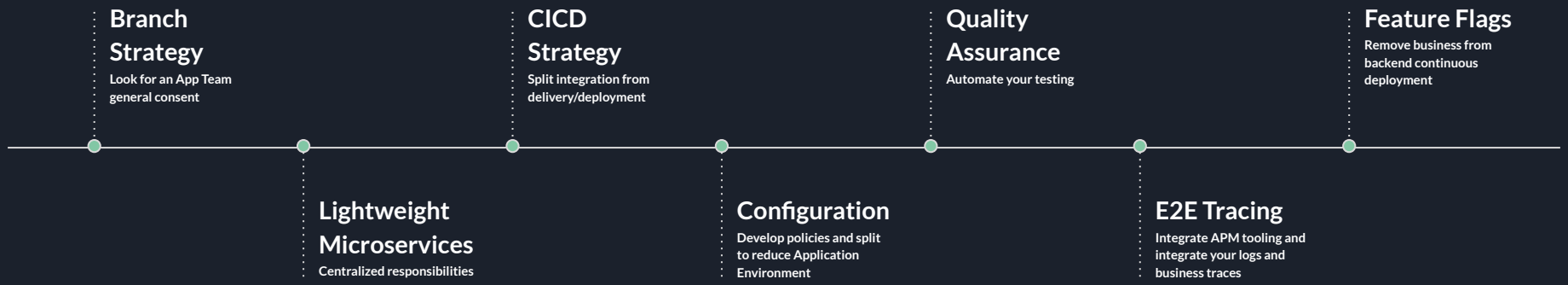
Schemas and data updates may require other data specialized team involvement attempting against the governance of the App team on the iteration of its solution.

- **Debugging**

High distributed logs due an architecture implicit bigger amount of components makes harder to drill down to errors root source.

DevOps

Planification guide



DevOps

Branch Strategy

- **Go with Trunk-Based-Development**

As a counterpart avoid getting into too complex other kind of branch strategies, the idea of adopting Microservices architecture is strongly related with smaller changes, retro-compatibility, automated validation and increasing production deployments frequence.

- **Use GIT Tags for versioning**

GIT Tags doesn't require commits then become simpler to automate the versioning. You'll be able to automate dashboards with environment Microservices versioning and several other processes such as environment nivelation, manual quality assurance, debugging your application, etc.

- **Use Build Validation features**

Put all existing mechanisms to prevent merging buggy code. Build Validation can trigger different pipelines where its succeed become part of the pull request required completion policies.

- **Version your API's**

Use same versions you included in your repo tags to correlate API's versioning into your API Manager proxy tools in order to offer those different versions. Then deploying multiple versions of those services is going to be required as backend part of your published API's.

DevOps

Lightweight Microservices

- **Remove unnecessary roles from Microservices**

Avoid software code duplication, keep your microservices small. Attend only for your methods and leave the rest where it belongs. Authentication, App roles or User accounts dedicated Microservices may be centralized roles within your IdP or Api Manager proxy tool. Extract JWT authentication information into headers and ship them to your Microservices.

- **Avoid develop your own Identity Provider (IdP)**

Use external ones, don’t deal with extremely complex authentication services by yourself, you’ll lose your time and you’ll probably build an insecure solution.

- **Use App Roles**

Another example is to avoid developing a full data model and its associated Microservices to offer roles in case your application needed. Get benefit from existing IdP roles features where the entire configuration rely over IdP as declarative objects. You can handle most of them using IaC in order to automate it and transfer keys to vaults where needed.

DevOps

CICD Strategy

- **Split Integration and deployment pipelines**

General idea is to decrease the amount of builds to minimal expression and trigger separate pipelines to carry with deployment role.

- **Promote Artifacts**

As another must, not just for Microservices, you have to deploy exactly the same you've test.

- **Use Templates**

For both Integration and Deployment pipelines, reduce duplication as much as possible and code your Branch Strategy conditions there. Is a channel to introduce changes on your CICD automation and massively impact all your Microservices.

- **Consider additional Triggers**

You can trigger your pipeline by just tagging a branch including another cross validations to avoid mistakes such as branches names, specific folder changes, or other builds executions to match all your expectations. Code your pipeline to reduce development and deployment efforts, you'll find a lot of improvement space there.

- **Automate your Schemas iterations**

Writing down into your repos your databases scripts and serves to iteration tools during deployment pipeline execution should be a mandatory requirement to deal with schemas updates. This extrapolates to other schema based services as cache, events, etc.

- **Automate your API's definition iterations**

Another mandatory requirement, extract API's definitions from code during integration, save then as artifacts and user them with corresponding tasks during deployment pashes.

DevOps

Configuration

- **Variables Policies**

Establish a well agreed App configuration variables policies and reinforce them with your team for its accomplishment.

- **Environment variables for App Properties**

At least main code languages allows to consume environment variables when loading as application properties. On promoting scenarios where same artifact is executed in different environments this feature is key.

- **Common infrastructure environment variables in Kubernetes**

Aiming for reducing duplication an accelerate infrastructure changes when needed you can create an automate another category of variables, those commons to each microservice and related with environment assets like cross services domains, dependencies base-url, dependencies credentials, etc. Then you'll be able to deploy infrastructure configuration changes and massively update your Microservices configuration.

Variables (Example): Environments and types	**Env. Affected**	**Env. Not Related**
Sensitive	Vault	Vault
Non-Sensitive	Library / Group	App properties file

DevOps

Quality Assurance

- **Understanding limitations**

Discuss with your App Team wath everybody understand and expect for Quality Assurance and positions related. Distinguish between business related features validations over Developers's technical user stories.

- **Code your Tests**

The more frequent and unattended desire for deployments the more testing automated needed. Consider a Tester Automation position in order to code for performance testing and integration testing. Include gate-policies for your code coverage and quality level.

- **Quality assurance vs user acceptance environments**

Understand the purpose of each environment, don't mix roles during QA and UAT phases. QA as near to developers and test automated as possible and UAT to dedicated business user validation if needed.

DevOps

E2E Tracing

- **Use an Application Performance Monitoring tool**

Besides its enormous contribution on tracing and profiling your Apps getting performance and behaviour information you can include additional traces as business events, logs and most of your apps dependencies, all of them correlated.

- **Real time Business monitoring from APM**

Do not access Microservices databases for Business operational monitoring. Use instead an external tool like APM to insert business events.

- **Limit the amount of traces ingestion**

Ingestion and indexing is a performance and economic cost you wants to efficiently delimit. Add daily caps into your ingestion service, sampling on higher environments and avoid duplication.

DevOps

Feature Flags

- **A different paradigm**

Have your business features and its user acceptance based on configuration flags distributed or centralized by Microservices on environment variables or configuration endpoints.

Use refinement meetings to convert business features to small Microservices oriented and high technically detailed User Stories.

Move finally to Continuous Deployment with Backend by promoting strongly validated changes to production as automated and unattended as possible.

DevOps

Further consideration

- **Canary deployments**

Add as many stages on continuous deployment as segments of users you have isolated and identified. Exponentially Increase the percentage of your features availability by continuously getting its operational feedback.

- **Configuration server**

Having an endpoint API with environment configuration will accelerate changes propagation and configuration governance.

- **Infrastructure as code**

Another paradigm change. By coding infrastructure you get benefits such as symmetrical environments, Infra disaster recovery, Infra environment testing, reduce iteration risks.

- **Center of Excellence**

“...Cross functional team who provides best practices, research, support or training for a focus area”
[*Wikipedia*]

Thank you for joining!

Globant

Create your way forward

(This humble presenter) **Gaston Cacheffo**
Tech Manager at Globant
gaston.cacheffo@globant.com

(Collaborator) **Luis Pulido**
Solution Architect at Globant
luis.pulido@globant.com