**Software Architecture**

CONTINUOUS ASSESSMENT ACTIVITY – CAA2

Nicolas D’Alessandro Calderon

**Presentation**

The CAA2 is the last activity of the course, which gathers some of the best practices in software development in distributed architectures. In particular, this activity works on key concepts about developing and putting distributed software into production, such as continuous delivery, the DevOps culture and the use of containers during deployment.

To elaborate this CAA, four exercises are formulated: 1) to evaluate one of the fundamentals principles of the DevOps culture; 2) to reason about the key role of tests in continuous delivery of quality software; 3) to evaluate a key aspect of containers and orchestration platforms; 4) to explain the construction of a “pipeline” for continuous delivery to test and deploy microservices implemented and tested in previous activities.

CAA2 covers the contents of the books "Microservices Patterns", "Continuous Delivery" and "The DevOps Handbook" (see Resources section for the chapters to be studied).

**Competences**

In this CAA2 the following competences of the Degree in Computer Engineering are worked on:

* Know how to build computer applications through development, integration and reuse techniques
* Know how to build computer applications through development, integration and reuse techniques.

**Objectives**

The objectives of CAA2 2 are:

* Recognize the factors of delivery, deployment and continuous integration.
* Interpret agile culture in software development.
* Understand the delivery, deployment and continuous integration pipelines.
* Explain the benefits and costs of containers and orchestration platforms.

Exercise 1

At the beginning of this activity we have proposed to you the following debate: "Discuss the basic objectives of a DevOps culture and the mechanisms, tools and/or strategies to achieve them". In this exercise we ask you to choose one of the objectives that you have discussed in the classroom Debate space to explain how you would apply it to the **Photo&Film4You** project.

“Hello everyone,

Since many topics have been already discussed, I’d like to share a personal experience from when as a Data Analyst I was forced to transition to AWS. This experience changed my personal and team’s culture when it comes to building the scalable data pipelines, which closely aligns with DevOps principles discussed here.

In my case, we were tasked to work with a sandbox environment that needed to be deployed to production using a CI/CD data pipeline. So, manage sandboxes requires to design structures that will not only work in test scenarios, but also to scale when deployed to production. This transition forced me to adopt a mindset of always thinking about scalability, automation, and monitoring from the beginning of the development process.

The process involves developing code in AWS services like Lambda or Glue and once the code is tested in a sandbox environment, it was deployed to production using a CI/CD pipeline. These pipelines include tools like TeamCity for build automation, ProGet for managing package artifacts, Octopus Deploy for deployment orchestration, and CloudFormation for defining the infrastructure as code. This pipeline ensures that the process is scalable, reliable, and automated for a smooth production deployment.

As a data analyst, it was easy to transition to the services that where data related like Lambda and Glue, but I found it very challenging to adapt to a CI/CD culture focused on these principles of scalability, automation, and deployment, especially since these concepts are more native to software developers than data professionals. In summary, even if at the beginning we found it very tedious and complicated, as a result, adopting a CI/CD culture is really worth it.

Some of the cultural principles that I learnt where:

- Thinking Scalable: When design data pipelines with infrastructure as code (IaC) you ensure consistency across environments.

- Automation: When building this CI/CD pipelines to automate deployment and avoid manual intervention, you reduce the risk of errors and improve the sped of the deployment.

- Collaboration: Working closely with DevOps/Infrastructure teams is the key to understand best practices, and also helps to understand and adopt a mindset of sharing responsibilities,

This experience has taught me that DevOps is not just for software developers. Whether you’re a data analyst, a business professional, or an operations specialist, adopting DevOps practices can result positive for across various roles.

Thanks for reading,

Nico.”

**I choose Scalability as the objective I would apply to Photo&Film4You**

In our project, the scalability is critical because we have concurrency issues and is also mentioned some periods of intensive usage that requires rapid development and deployment cycles.

By applying scalability principles, **Photo&Film4You** will be able to handle spikes in usage, keep the performance when high concurrency, and also ensure the quick deployment of new features.

So, based on my answer to the discussion in the forum, the ***scalability*** principle can be applied following, for example, this steps :

1. **Infrastructure as Code** for consistency across dev. environments

For example, as mentioned in my post, tools like *CloudFormation* that define the infrastructure in the cloud environment can be applied here using tools like Terraform or AWS CDK. This will ensure that the MySQL databases and the microservices for **Photo&Film4You** keep the consistency of the environments such as development, testing, and production.

Since we have the deployment written in code (including the databases services, permissions, etc), the system can scale horizontally during peak periods without manual intervention.

2. **Automation** with CI/CD pipelines

Implement CI/CD pipelines to automate the deployment of the microservices, including the Product Catalog, Rent, and User services. Tools like Jenkins or GitHub Actions may be used for building, test, and deploy the changes automatically, so we will reduce errors and improving the speed of updates.

For example, when introducing a new functionality like the digital sessions or updating the product descriptions, this automated pipeline will ensure the integration with the current system.

3. **Collaboration** for monitoring and alerting

The collaboration with the DevOps team in the **Photo&Film4You** project is essential for the set up and the monitoring (we can use tools like Prometheus or Datadog). This will ensure that we can track the system performance in real time and automate alerts for the already mentioned periods of high resource usage, which will enable proactive scaling.

**How to apply this to Photo&Film4You and expected outcome:**

**Horizontal Scaling for High Concurrency**

Using containerization (Docker) and orchestration (Kubernetes) to run microservices on distributed nodes. As we have in the project stub given, microservices like *ProductCatalog* and *Notification* can scale independently based on traffic demands, and avoid bottlenecks.

**Auto-Scaling Cloud Resources**

We can also benefit from auto scaling groups for servers and databases that during high-demand periods will provide additional instances of the Rent service for example or database replicas can be created and dynamically and scaled down during low demand hours.

Build

Pipeline

Deployment Pipeline

Prod environment

Developer commit code

Automated tests

Exercise 2

In PRAC2 you carried out the implementation of Photo&Film4You's Product Catalog microservice while in PRAC3 you developed a set of unit tests, using JUnit, Spring Boot Test, Mockito and AssertJ on this microservice. Briefly answer the following questions:

1. The use of different types of tests provides a certain level of assurance for the operation of microservices at various levels, but we lack an understanding of the quality of the generated code. Analyze static and dynamic code analysis techniques in CI/CD to assess and continuously improve the code quality of developed microservices.
2. Consider adopting a code quality analysis system, such as SonarQube. Explain the benefits it would bring.

**1. Static and Dynamic code**

**Static:** These are tools that help in the process of analyzing the code but without running it. PMD and SonarQube for example are tools that helps to check **syntax errors, code smellls (complex functions or unused variables), coding standars and security vulnerabilities.**

These static code techniques helps to ensure that the code is clean and consistent, and it also heelps to identify bugs before they reach the testing production, what will finally reduce the technichal debt

**Dynamic:** This is the process of analyzing the code when it is running. Tools like JUnit and Mockito test how the microservice is performing in reeal conditions **by simulating request to endpoints, testing the business logic and the database operation, mesuring memory usage, response time, etc.**

These Dynamic techniques helps to understand if the code is behaving as expected in different scenarios, and also to detect integration issues and validates that the microservice is meeting the functional requirements.

**2. Benefits of adopting SonarQube**

SonarQube will help to improve the overall code quality because is excellent for eraly detection of issues since it scans the entire codebase for bugs and find vulnerabilities and refactor needs, so it helps the teams to fix the problem before they scalate.

On the other hand, SonarQube will also be benficial because it can be integrated with CII/CD pipelines ensuring the quality oof the entire project.

These tool offers the posibility of customize rules, support different languages and also allows teams to set their own coding standards. In our **Photo&Film4You** project for example, we may cretae specific rules for *SpringBoot* or *MySQL* best practices.

This tool also provides the possibility of creating reports and dashboards that will make easier for the developers to understand the issues and act on it.

Finally, adopting this tool will increase team collaboration, improve the code mainteinability and detect vulnerabilities like SQL injections and invalid inputs thatt may compromise the microservice.

Inthe **Photo&Film4You** pipeline, using SonarQube or similar tools for the Product Catalog and other microservices, can ensure that the code is clean, secure, and efficient every time a change is pushed.

Exercise 3

Choose a possible deployment strategy for the Photo&Film4You Product Catalog microservice.

Describe the strategy you choose and list its advantages and disadvantages. Specify which deployment environment enables achieving both vertical scalability (within the same machine) and horizontal scalability (across multiple machines). Propose a global monitoring solution for the service, detailing the tools and techniques that can be used to ensure reliability, performance, and early detection of issues

Based on the learning material, I will choose the **Blue-Green Deployment** strategy described in the book, because it seems ideal for the **Photo&Film4You** project.

This strategy could be appropriate because ensures minimal downtime and safe rollbacks during deployments.

The basic idea is to maintain two identical environments, the **Blue** (current prod) and the **Green** (the new version). The deployment of the microservice version will be done to the green environment where we will run the tests to ensure everything is working correctly. When ready, we will switch the traffic from the Blue to Green and if any issue occurs, we will quickly switch back to Blue.

The advantages of these approach are mainly **Zero downtime**, which means that we keep the user experience with no interruption. It is also optimal for Safe rollbacks, meaning that if the new version fails, we can instantly switch back. Finally, other advantage is that the Green environment can be tested isolated, without affecting the Blue.

Regarding disadvantages, it is mentioned that this strategy is costly due to the need of two environments. Also, migrating data between environments sometimes results in complexity process.

**Scalability**:

To achieve the two proposed **Vertical** and **Horizontal** scalability, we will use a containerized platform like Kubernetes. We will be able then to add resources like CPU or RAM to existing containers (Vertical Scalability) and also increase the number of containers replicas to distribute the load. (Horizontal Scalability).

On the other hand, Kubernetes also supports auto-scaling, which benefits I have already described in Exercise 1 answer. This will allow adjust the resources automatically based on the demand.

**Monitoring Solution**

Regarding this, we may need to focus on the early detection of issues, so a tool like *Prometheus* can be a good option. With this tool we will be able to collect metrics like CPU, memory and response times and send alerts via its **Alert-manager** component if something goes wrong.

The techniques that we will use, are the mentioned Smoke tests, aimed at verifying if the systems run correctly after deployment. We will also perform regular **Health Checks** to ping endpoints and check its status. Finally, the **Log Analysis** technique to register and monitor logs for errors.

With this monitoring strategy we will be able to early detect the issues as well as track the performance in real-time by having a centralized dashboard with the metrics that will indicate the system health.

Exercise 4

Explain in a generic way what should be done in the Photo&Film4You project to achieve Continuous Deployment and indicate how it would be done if we use GitLab as CI/CD platform.

Analyze the possibilities of integrating GitLab CI/CD with other systems, for example, to receive notifications of the pipeline results in a system like Slack (<https://slack.com/>)

**Note**: You are not required to specify the specific GitLab commands, we only ask for an overview of the process and the steps involved in the process.

These can be the generic steps for **Continuous Deployment** in **Photo&Film4You:**

**1. Version control:**

The code is stored in a GitLab repository.

The Developers push the changes to specific branches.

**2. Automated testing:**

When the code is pushed, GitLab CI/CD will run automated tests.

The tests will ensure that the new code is not breaking any existing features.

**3. Build:**

If the tests pass, then the application is built into a Docker image.

The image will contain the updated microservice and it will be ready to run.

**4. Deploy to Staging:**

The new Docker image will be deployed to a staging environment.

The idea of the staging is to imitate the prod env for further tests and validations.

**5. Automatic deployment to Production:**

If the test in staging is successful, then it will be deployed to production

automatically. Kubernetes will be used to handle the deployments

and orchestrate the Docker containers.

**Integrating GitLab CI/CD with Slack or similar tools:**

Integrating Gitlab CI/CD with Slack will serve for real-time notifications, so all the team is always informed about the results of any triggered process in the pipeline

GitLab can send success or failure results notification to a Slack channel.

*Example:* If a test fails, the team knows immediately and can take action to fix the issue.

So, in summary we will have Real-time updates to keep the team informed ending up in a faster response to issues that improves reliability.