DevOps Fundamentals

About me

- Pablo Maldonado, PhD.
- Trainer / Lecturer.
- Based in Prague, Czechia.

About you

- Name and what do you do?
- Experience with:
  - Git
  - Docker
  - DevOps in general?
- 1-2 concrete things you would like to know in the end of this course.

Schedule

- 9:30-12:30 Morning session.
- 13:30-16:30 Afternoon session.
- Small breaks in between.

Introduction to DevOps

Getting Started with DevOps

- Term introduced in 2007-2009 by Patrick Debois, Gene Kin and John Willis.
- Contraction of **Development (Dev)** and **Operations (Ops)**.
- **Aim:** Reduce bareers between developers (who want to innovate faster) and operations (who want to guarantee stability).

Key Elements

- More frequent application deployments with **CI/CD**.
- Implementation and automation of unitary and integration tests (**BDD/TDD**).
- Implementation of a means of collecting feedback from users.
- Monitoring applications and infrastructure.

The 3 Axes of the DevOps movement

- Culture of collaboration.
- Processes.
- Tools.

The 3 Axes: Culture of collaboration

- Teams are no longer separated by silos.
- Instead, multidisciplinary teams that have the same objective: deliver added value as quickly as possible.

The 3 Axes: Processes

- Agile methodologies with iterative phases for better functionality and rapid feedback.
- Agile mindset for *both* development and deployment.
- The DevOps process is divided in several phases:
    - Planning and prioritization of functionalities.
    - Development.
    - Continuous integration and delivery.
    - Continuous deployment.
    - Continuous monitoring.

The 3 Axes: Tools

- Choice of tools and products that facilitate collaboration.
- Tools must be usable by everyone in the team, regardless of their specialization.
- Developers need to integrate with monitoring and security tools used by Ops teams.
- Ops must automate the creation and updating of infrastructure and integrate the code into a code manager (**Infrastructure as Code**).

Benefits of DevOps

- Better collaboration and communication in teams.
- Better performance and end user satisfaction.
- Reduced infrastructure costs with IaC.
- Significant time saved with iterative cycles that reduce application errors and automation tools that reduce manual tasks.

CI/CD

CI/CD

- Continuous integration (CI)
- Continuous delivery (CD)
- Continuous deployment

Continuous Integration

- Software development practice where members of a team integrate their work frequently.
- Each integration is verified by an automated build (including test) to detect integration errors quickly.
- CI consists then of two parts:
    - Source code manager (Git/SVN).
    - Automatic build server/ CI server (Jenkins, Gitlab CI, Github Actions, Travis CI)

CI Workflow

- Each team member works on the application code daily, iteratively and incrementally.
- Each task/feature must be partitioned from other developments with the use of branches.
- Each member archives/commits their code in small trunks that can be easily fixed.
- The new code is integrated to the rest of the app.
- The CI server retrieves the code and does the following:
  - Build the application package.
  - Perform unit tests.

Continuous Delivery

- After CI, the application is deployed in a non-production environment, called **staging**.
- The code package generated after CI must be the same (binaries, DDL, JAR). Only changes on configuration files are allowed!
- This is important because it is the only guarantee that the version deployed to production would be the same as in staging.
- Deployment to production can be trigerred automatically or manually, after acceptance of testing users.

Continuous Deployment

- End-to-end automation of the CI/CD pipeline.
- Rarely implemented in enterprises.
- Can be implemented in two main ways:
  - Feature flags (on/off features directly in production).
  - Blue-green deployment: two production environments, deployment goes first to blue and then to green.

Infrastructure as Code (IaC)

Understanding IaC practices

- IaC is the process of writing the code of the provisioning and configuration steps of infrastructure components to automate its deployment in a repeatable and consistent manner.
- Declarative instead of imperative description of the infrastructure setup.

Benefits

- The standardization of infrastructure configuration reduces the risk of error.
- The code that describes the infrastructure is versioned and controlled in a source code manager.
- Deployments that make infrastructure changes are faster and more efficient.

Benefits (cont.)

- Better management, control, and a reduction in infrastructure costs.
- Easy for developers and testers to recreate self-service, ephemeral environments.
- **Tools:** Terraform, Ansible, Chef, Puppet and others*.

Example: Terraform

```
resource "azurerm_resource_group" "myrg" {
name = "MyAppResourceGroup"
location = "West Europe"
tags = {
environment = "Mydemo"
}
}
```

Example: Ansible

```
---
- hosts: all
tasks:
- name: stop nginx
service:
name: nginx
state: stopped
- name: check nginx is not installed
apt: name=nginx state=absent
```

Containers

- Containerization means that an application will be deployed in **containers** instead of VMs.
- Containers are built from pre-made **images** and defined via **Dockerfile**.

```
FROM ubuntu
RUN apt-get update
RUN apt-get install -y nginx
ENTRYPOINT ["/usr/sbin/nginx","-g","daemon off;"]
EXPOSE 80
```

Configuration and deployment in Kubernetes

- Kubernetes is a container orchestrator.
- Deploys containers, the network architecture (load balancer, ports) and volume management.
- They are defined completly in YAML specification files.

Kubernetes YAML example

```
apiVersion: apps/v1
kind: Deployment
spec:
        replicas: 2
        labels:
                app: nginx
spec:
        containers:
        - name: nginx
                image: nginx:1.7.9
        ports:
        - containerPort: 80
```