T7 - Network & Sys Admin

T-NSA-700

devOps

bringing agility to the product development cycle

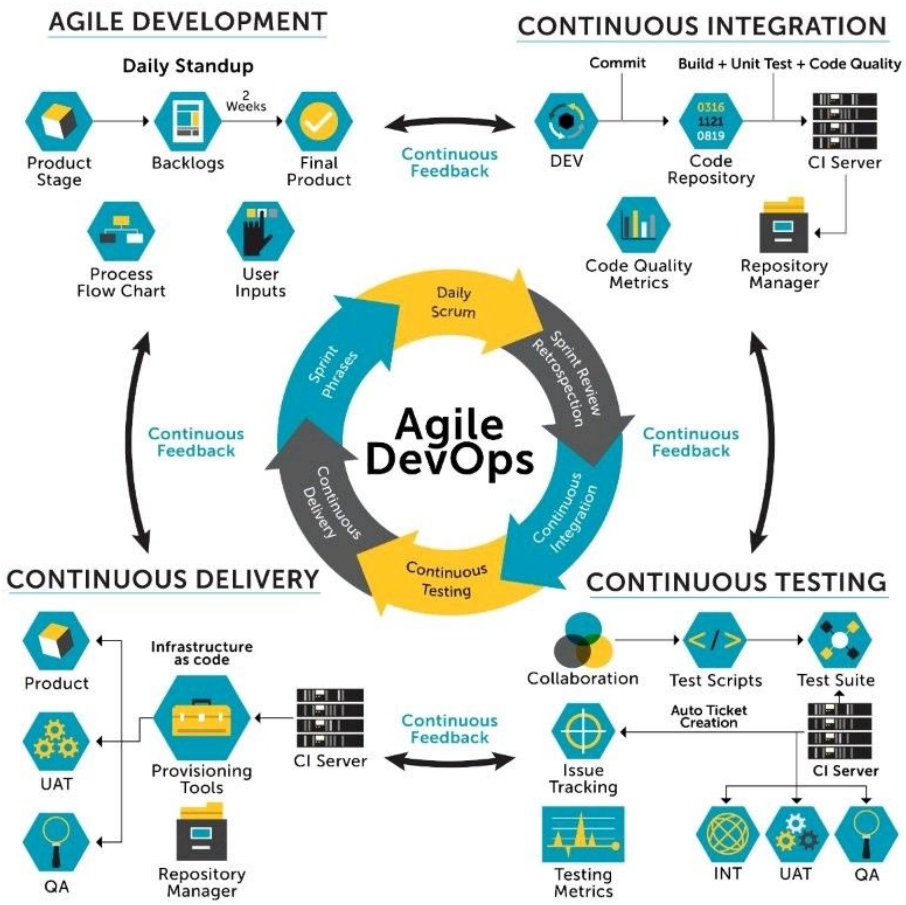
1.6.1

devOps

binary name: devops_*$firstname.$lastname*.zip
delivery method: Moodle

- The totality of your source files, except all useless files (binary, temp files, obj files,...), must be included in your delivery.
- All the bonus files (including a potential specific Makefile) should be in a directory named *bonus*.

To fit the market and be more agile, your company decided it needs speeding up its delivery cycle. There was a big, ambitious plan for that:



> *DevOps adoption means that you are willing to change fast, develop fast, test fast, fail fast, recover fast, learn fast and push the product/features fast to the market.*
>
> *Pavan BELAGATTI*

The first step was to use *Scrum* as a project management method. The next bottleneck to be copped with is deployment on production.

Initially, the goal is to create a shared hosting platform in which developers will benefit from convenient and secure tools to develop and deploy applications.

The company also wants to be able to easily deploy servers in this new infrastructure, and automate the deployment of the necessary tools on it.

It has therefore drawn up specifications for the technologies that it wishes to see implemented within the framework of this infrastructure, leaving technological choices at your discretion and a list of the use cases that it wants to see presented during the platform demonstration.

Despite your recommendations, **this company forbids you any use of Docker**, because "the security is not guaranteed". You'll have to stick to it …

Platform/Infrastructure layer

Let's start by creating the basic infrastructure on which the demonstration application will be deployed. A basic technological determination has therefore been done, representing the expected infrastructure:

1. Setting up a Gitlab server. It can be done manually, without Ansible. Docker is permitted here.
2. The installation of services and their dependencies (Nginx, php-fpm, MySQL…) must be done with *Ansible*. The configurations will be separated into appropriate *roles*, to be reusable across your different *playbooks*.

Application layer

Two applications have been provided to demonstrate the operation of the infrastructure:

- a *backend* **Laravel**
- a *frontend* **Angular**

For each of these applications, you will need to provide a pipeline that builds the application via:

- A *build* step containing the build of the application, and creating an artifact, containing all the dependencies necessary for running this application into production

- A *test* step running the unit tests
- A *deploy* step executing an *Playbook Ansible*.

build and *test* steps must run automatically on push. *deploy* will be executed on demand.
For each of these applications, you will also need to provide the *Ansible* role to deploy it (the **CD** part) ; these *playbooks* will deploy and install the artifacts of your applications to make them accessible.

During the deployment, the *down-time* should be as short as possible, 5 seconds seem largely sufficient!

The database schema is set to evolve. You will need to provide an automatic migration mechanism.

In case of regression, we would like to revert to the previous release in less than 3 minutes.

You should check on Google how to deploy Angular and Laravel applications!

Deliverable

You must deliver a Gitlab server responsible for *CI* and *CD*, as well as 3 blank Debian servers (for MySQL, Frontend and Backend).

You must submit to Gandalf a zip file, containing your repositories created on Gitlab. It must bring, at least, your CI/CD pipelines and Ansible recipes.
You will need to be able to perform the following demonstration:

1. Prepare the infrastructure layer and configure the machines with *Ansible* from a terminal

2. CI: build, test and package your appliccations on Gitlab
3. CD: launch the deployment of each application with *Ansible*, from Gitlab
4. Access each application

Keep you virtual machines lightweight: no need for window manager, GUI, Plesk.…

Security should not be overlooked, no password will be *visible/readable* in your git repository (nor in its history!).

Bonus

To go further, and keep having fun, you could potentially consider:

- add end to end automated tests in pipeline
- setup a branching strategy: "git flow" or "Github flow"
- a staging platform
- the use of public DNS and automatic generation of SSL certificates
- deployment on VPS (OVH, Scaleway …)
- an infrastructure in high availability (MySQL in master/slave, load balancer …)
- provisioning cloud resources with Terraform
- building a VM template using Packer
- a database backup
- log aggregation
- monitoring
- other creative things …