Spring - Mar 16

First, when start working, we should have a Laptop which is our company’s property

- local Env - local environment

First day → login source control management(SCM)/git and SVN(old) are most commonly use

- Gitlab
- bitbucket

codes from SCM to local env: check-in / check-out

- To avoid two people modifying the same file at the same time you would "check it out" before starting modifications and "check it in" when your modifications were done.
-

> Think of it as arriving at a place to stay (a hotel, a resort, etc):
>
> - the very *first* thing you do (when you arrive) is to **`checkin`**.
> - the very *last* thing you do (when you leave) is to **`checkout`**.
>
> A similar SCM concept applies when you want to apply changes to software components ... except that ***it applies the other way around***:
>
> - the very *first* thing you do (when you start) is to **`checkout`** (or think of it like borrowing it).
> - the very *last* thing you do (when you finish) is to **`checkin`** (or think of it like giving it back).

Note: this applies to centralized systems (such as the ones used in mainframe environments ...). In systems such as `git` the "*`checkout`*" concept has a completely different meaning (which IMO is also why in those systems there is hardly any confusion about both concepts).

When all codes are ready, we need to deploy to servers

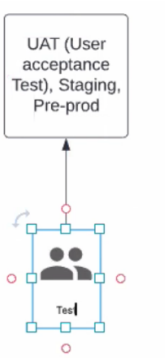
Instead of calling them servers, we call them environment

- The final environment: user to use - Production Env(real deal, real user, real money, real business)
    - we can directly push codes to Production Env(risky), local env ≠ PE
        - local env could be separate codes of all program

Before PE, we have lots of environment for testing

UAT (User acceptance Test), Staging, Pre-prod(Pre-production)

- propose:
  - conduct user acceptance test
  - rehearsal
- users are not real users who are testers



What the UAT have, What Production has

We could never see production env, cuz like we build a house but we never have permission to use it.

Test/Integration evn

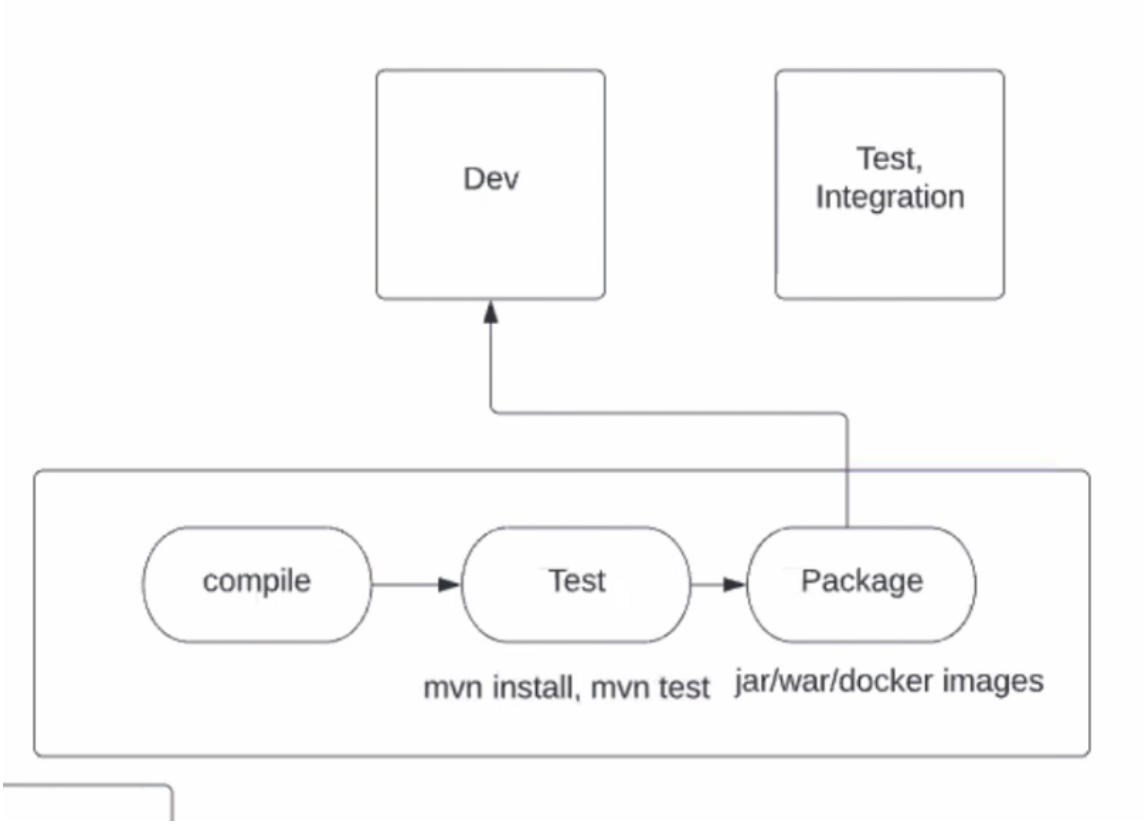
Dev - develops environment

- Dev test UAT and Production

plus local - we have 5 environments

flow:

- write files local → test
  - compile → test → Package
  - Package → copy → Dev
  -



- deploy process
- DevOops people's job

Why we need dev env?

- we write codes in local, it's working in local, so we want to try whether it's working in servers
- it's whole purpose for dev env, help us to verify the codes that's working on the local, still working on the server
- Java is platform independent → why we need to test different env
  - Java is independent but our codes are not
  - Our codes could have lots of env independent
    - like some paths is locally "c:/windows/xxx" which does not work on Linux servers
  - also has network → Dev and local env have different networks(access/security/policy)

each env has its own DB

Who will use the Test/Integration?

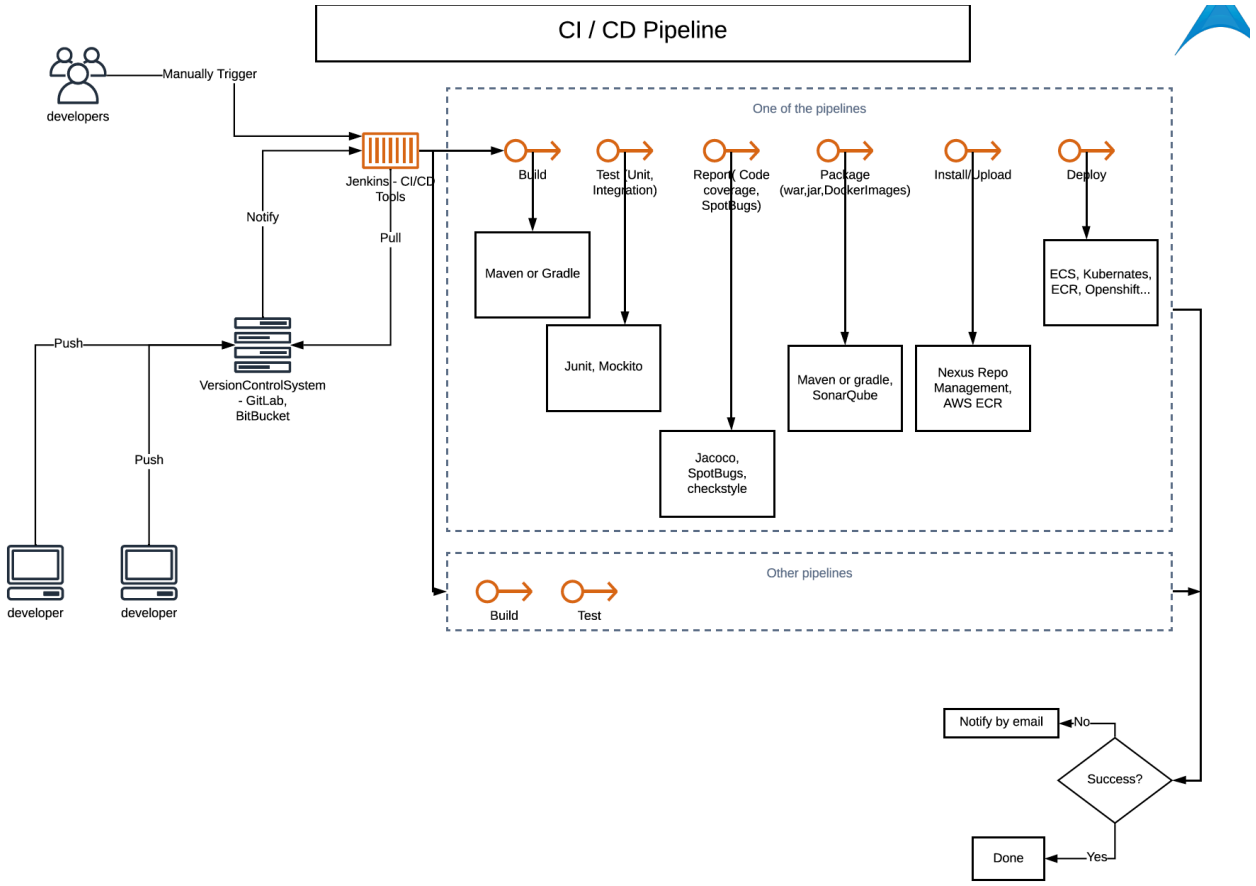
- each sprint will re-deploy the test env to apply its new features
- we deploy it and the testers who we call QAs will come and test our env, and the env can be used in demo to the client
- Test could be changed again and again(agile)
- release → the codes are tested well can be ready to use for real users
    - deploy → UAT → Production
- during the middle, there is codes freezing, which the codes prepare to deployed, and make sure no one can change it

Intro new tech:

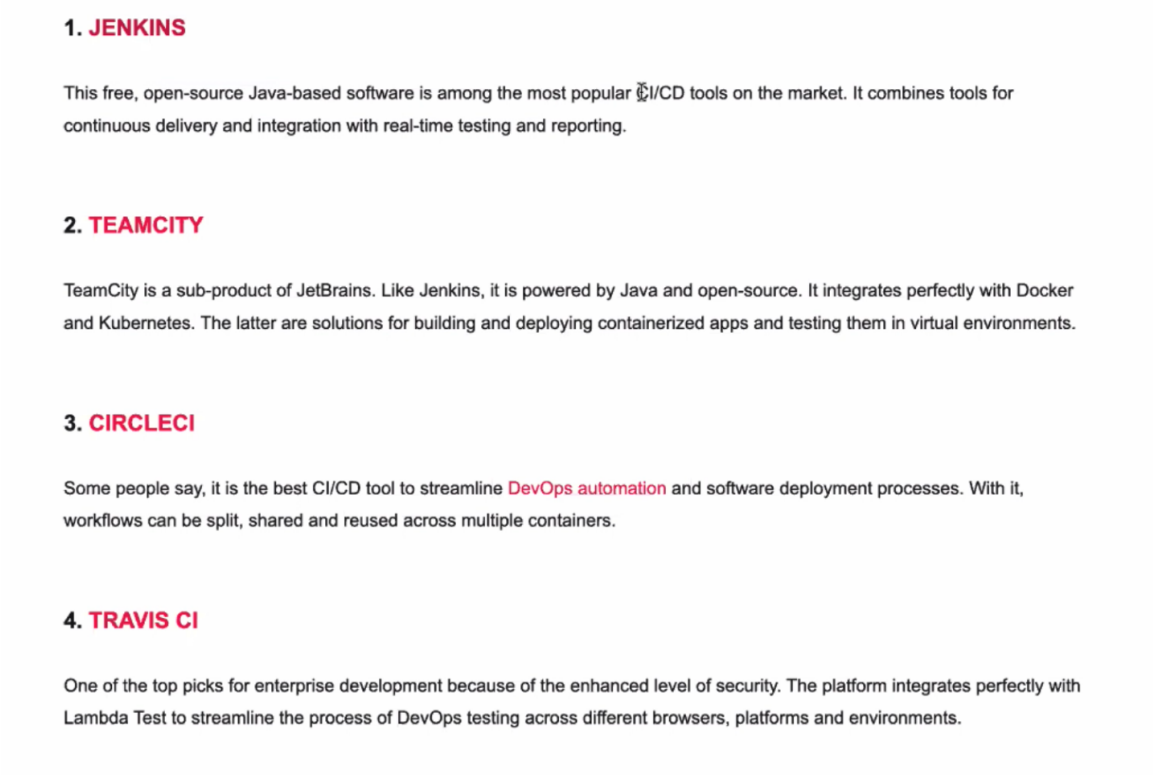
Continuous Integration / Delivery or Deployment (**CI/CD**)

- delivery means the codes are always ready for deployment
- integration means the code is always be integrated all the time and well tested

Jenkins -CI/CD



- push the codes to git
- git will notify the Jenkins server(trigger)
- Jenkins pulls codes from git and start the pipeline
- pipeline is configurable depend on different teams
  - it can have multiple pipelines (the picture sh)
  - build → test → report → package → install/upload → deploy
    - upload does not just upload to server directly
      - there are separate places we call them repository(repositories) / artifact repository
      - the repository will save all package files/ and their history version and so on(like git/ version control)



1. JENKINS

This free, open-source Java-based software is among the most popular CI/CD tools on the market. It combines tools for continuous delivery and integration with real-time testing and reporting.

2. TEAMCITY

TeamCity is a sub-product of JetBrains. Like Jenkins, it is powered by Java and open-source. It integrates perfectly with Docker and Kubernetes. The latter are solutions for building and deploying containerized apps and testing them in virtual environments.

3. CIRCLECI

Some people say, it is the best CI/CD tool to streamline DevOps automation and software deployment processes. With it, workflows can be split, shared and reused across multiple containers.

4. TRAVIS CI

One of the top picks for enterprise development because of the enhanced level of security. The platform integrates perfectly with Lambda Test to streamline the process of DevOps testing across different browsers, platforms and environments.

CI/CD tools

ATLASSIAN → Jira issue tracking system

CodePipeline: AWS solution for CI/CD

Jenkins (00:51:44)

Agile **methodology**

mean whole team follows agile way: fast develop, deploy, delivery...

we plan partially, we design partially, we develop partially, we deploy partially

we shows the demo to client whether it is they want. if yes, we move on, if not we go back(feedback, change)

- Scrum
- Kanban

Backlog (1:13:28)