Deployment pipeline

Agenda

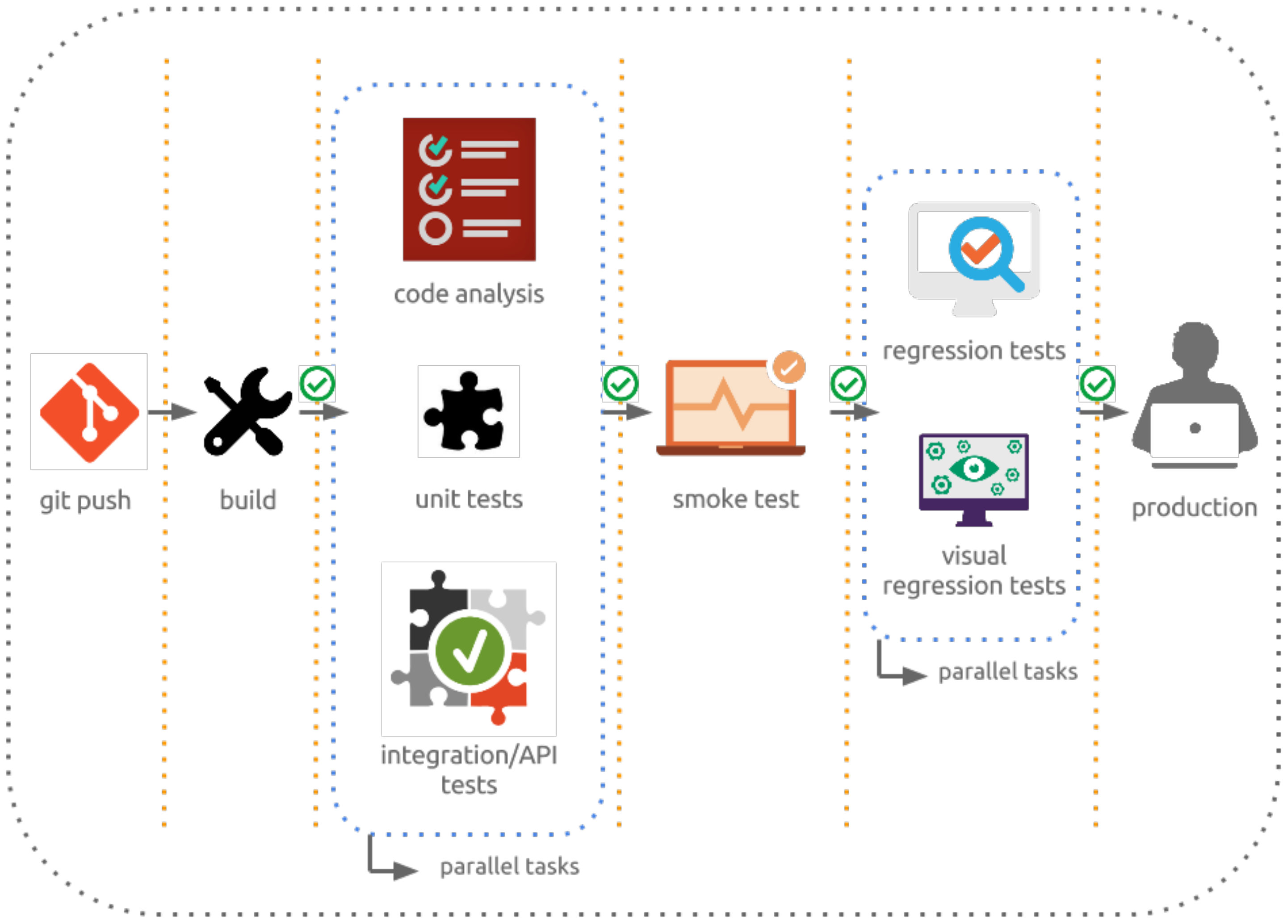
- What is a deployment pipeline
- Delivery vs deployment
- Deployment environments

Deployment: definitions

- Bass et. al.: "***Set of steps** taken between a developer commiting code and the code actually being promoted into normal production while ensuring high quality*." [1], page 80.

- Humble et. al.: "*Automated manifestation of your **process** for getting software from version control into the hands of your users*". [8], page 106.

Example



https://medium.com/the-making-of-whereby/what-is-a-deployment-pipeline-and-how-it-helps-software-development-teams-6cb29917ceea https://www.guru99.com/smoke-testing.html

Remark: smoke testing

- smoke testing (also confidence testing, sanity testing, build verification test (BVT) and build acceptance test) is preliminary testing to reveal simple failures severe enough to, for example, reject a prospective software release.
- Smoke tests are a subset of test cases that cover the most important functionality of a component or system, used to aid assessment of whether main functions of the software appear to work correctly.
- When used to determine if a computer program should be subjected to further, more fine-grained testing, a smoke test may be called an intake test.
- Alternatively, it is a set of tests run on each new build of a product to verify that the build is testable before the build is released into the hands of the test team.
- In the DevOps paradigm, use of a BVT step is one hallmark of the continuous integration maturity stage

Process / Set of steps

1. Developer designs, creates and tests code on local *environment*.
2. Developer checks code* into a Version Control System (VCS).
3. The system** is built
4. A series of unit and integration tests are run.
5. After passing the tests, the system is promoted to a staging *environment* where it undergoes more tests (e.g. performance, security, and user acceptance tests).
6. The system is finally promoted to normal production***.

(*) Also tests and any other artefacts.
(**) Referred to as **candidate**.
(***) The tests do not necessarily stop.

The process

1. Developer designs, creates and tests code on local environment.

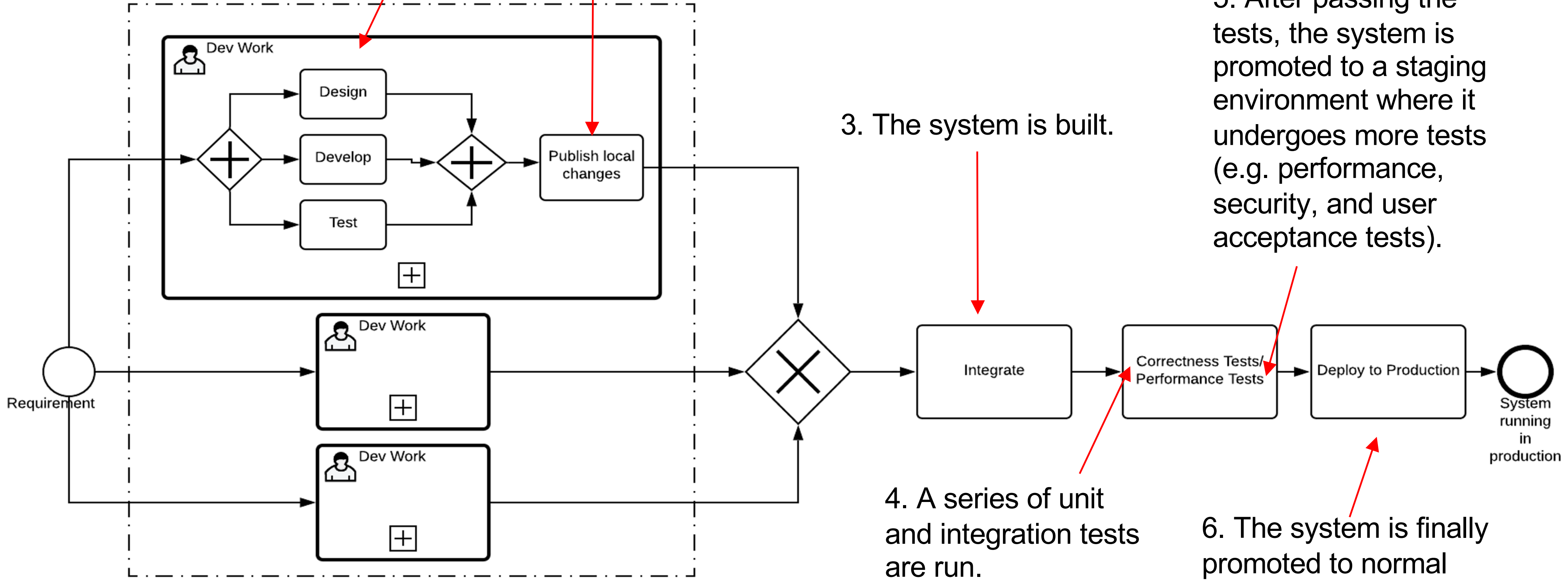
2. Developer checks code into a Version Control System (VCS).

3. The system is built.

4. A series of unit and integration tests are run.

5. After passing the tests, the system is promoted to a staging environment where it undergoes more tests (e.g. performance, security, and user acceptance tests).

6. The system is finally promoted to normal production



Refining Dev

- Retrieve version of the code to modify from VCS.
- Create new branch to keep the code being modified.
- Interact with IDE to make modifications
- Build the code using an environment* loaded with the dependencies (OS, libraries, and dependent modules) required.
- Test the code (with existing and new test cases)**.
- Perform static analysis on the code.
- Peer review the code.
- Commit and **push** the modified code in the VCS***.

(*) The IDE has to use this environment as target for its activities

(**) Test cases should be in the VCS.

(***) Push modifications to the branch from where you retrieve the code.

Refining Integration

- Build an executable version of the system (i.e. creates candidate).
  - Includes loading, and compiling the complete source code of the system.
    - Check the code can be compiled and linked with dependencies -> useful feedback
    - Package the system just built
- Run unit tests
  - Run very fast -> useful feedback
- Run integration tests
  - focus on functional correctness / only those that run somehow fast -> useful feedback
- Remarks:
  - These steps are executed in a particular -> integration environment
  - The creation of the environment and the execution of the steps within it is done by the CI server.

Refining Test

- Test the user acceptance & qualities of the candidate.
  - Functional testing (based on predefined scenarios - ideal case)
  - Performance efficiency (time behaviour/ Resource utilization/ Capacity)
  - Security
  - Other “ilities” of interest (NFR: Non Functional Requirements –see the related slides later)
- Place the candidate on a repository for deployment to production.
- Remarks:
  - An environment very close to production is required to perform the tests.
  - The configuration parameters of this environment should be the same as the production environment, except for:
    - Separated database (but with relevant data)
    - Different credentials
    - Mock external services the system writes to

Refining Deploy

- Place the candidate into production environment
  - Replace old version for new one
- Remarks:
  - Problem during the release process
    - Mitigated by rehearsing the process
      - Using the same process than in other environments
  - Defect in the new version of the system
    - Mitigated by keeping the previous version of the system available while the new version is released.
      - Big deal -> Data

Environment

- What is it, actually?
  - A **place** where to perform certain activities
    - mainly related to testing (but not only -e.g. compiling, and packaging)
- Requirements
  - Isolated from other environments (please check dependencies)
  - Should mirror (to same extent) the production environment
  - The elements that are promoted from one environment to the next one should be recorded for traceability and repeatability.
- Property: finite lifetime
  - Created, used, destroyed.

Different environments in the pipeline

- Development environment -> development activities
- Integration environment -> integration activities
- Staging environment * -> test activities
- Production environment -> operation activities, namely where the system is actually used

(*) Different organisations may have additional Staging environments defined.

DELIVERY ≠ DEPLOYMENT

DELIVERY		DEPLOYMENT
Test		Test
Staging		Staging
MANUAL	DIFFERENCE	AUTOMATIC
Production		Production

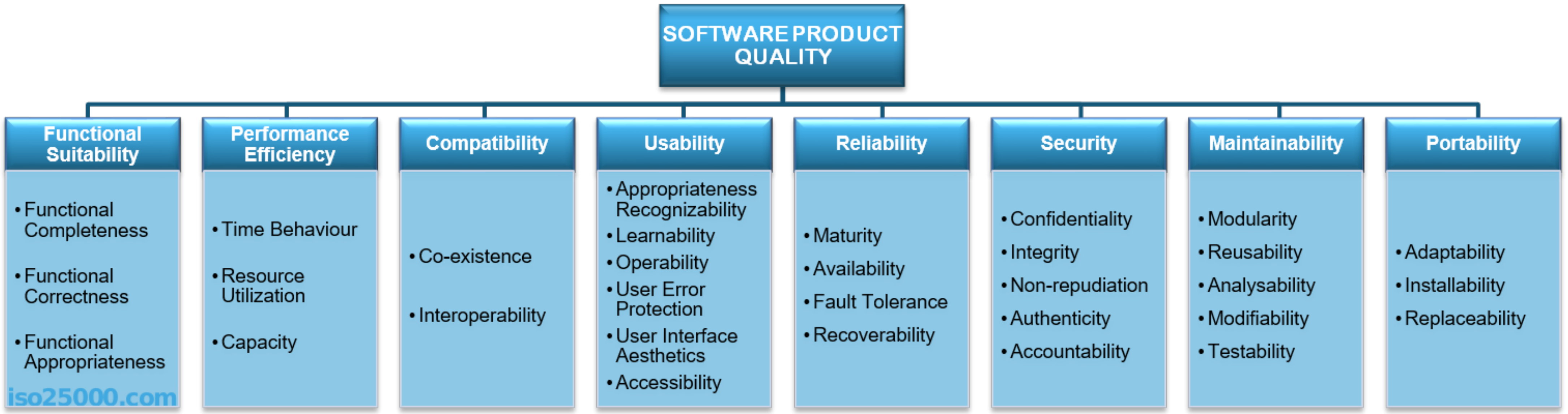
Functional requirements (FR) for the pipeline

- (FR1) Team members can work on different versions of the system concurrently.
- (FR2) Code developed by one team member does not overwrite code developed by others.
- (FR3) Code produced by one team can be integrated with code produced by other teams.
- (FR4) Code is the same during different stages.
- (FR5) Different environments serve different purposes.
- (FR6) Different environments are isolated from each other.
- (FR7) A deployment can be easily rolled back if there is a problem.

Non-Functional requirements (NFR)

- NFRs: **quality** concerns other than those focus on the functionalities
- The **quality** of a system is "*the **degree** to which the system **satisfies the stated and implied needs** of its various stakeholders, and thus provides value*." (ISO/IEC 25010:2011(E).
- Quality of a system -> "**ility**" of a system
  - Reliability,
  - Compatibility,
  - Maintainability,
  - Portability ,...
- Targeted "ilities"
  - ISO/IEC 25010 - Systems and software engineering - Systems and software Quality Requirements and Evaluation (SQuaRE)

Software product quality ISO std 25010

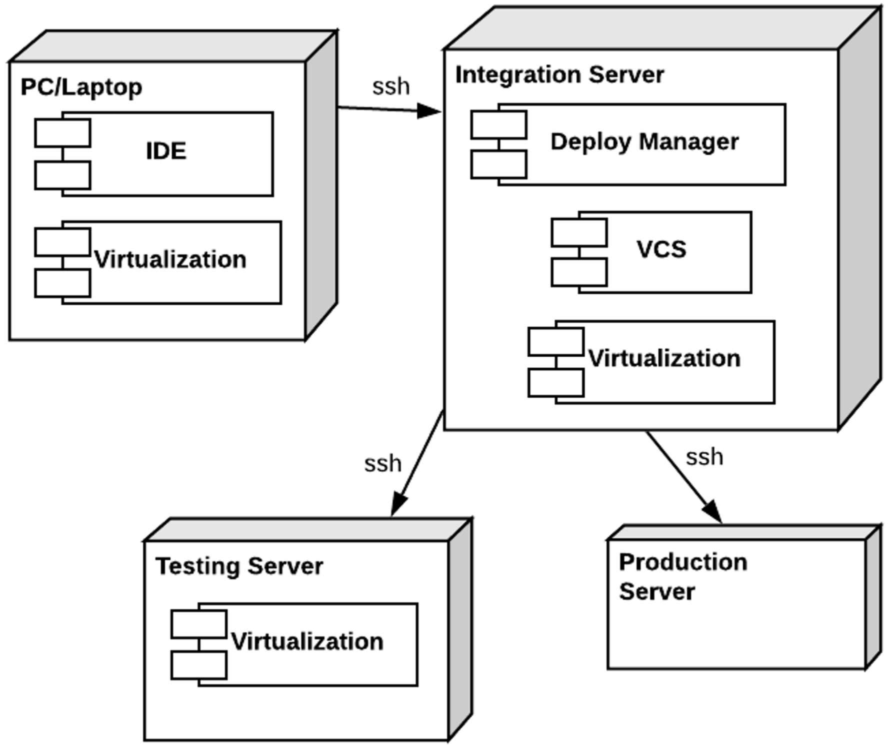


DevOps environment

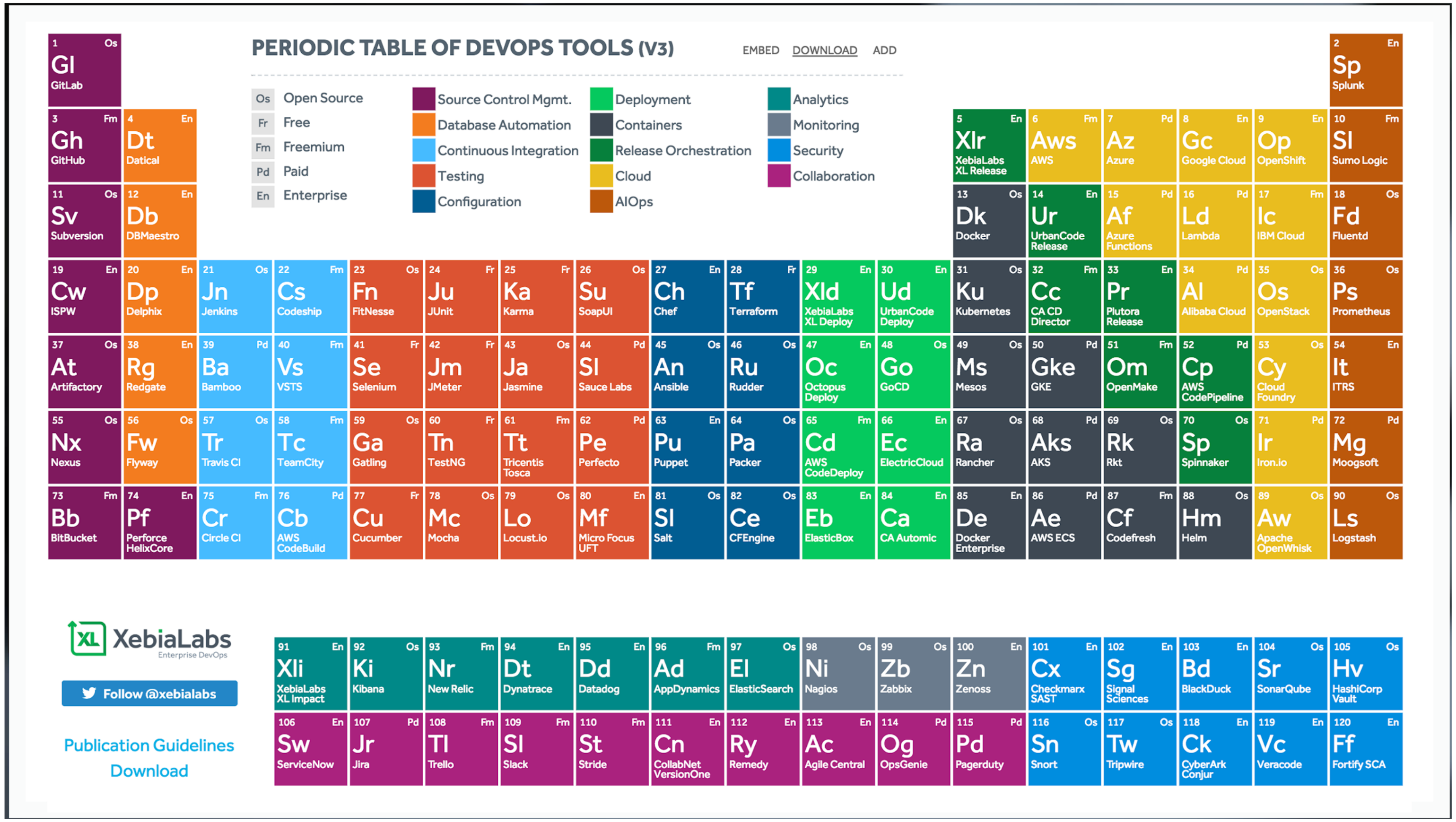
- Collections of services and resources
- Goal:
  - let the system to work properly
  - let the organisation to succeed in making such a system and, eventually, achieve its release.
- Synonymous
  - Infrastructure
  - Delivery ecosystem
- Remark: DevOps environment may include more things than those required to support a pipeline
  - e.g. monitoring services

Example of a DevOps environment architecture

- (FR1) Team members can work on different versions of the system concurrently. **(VCS)**
- (FR2) Code developed by one team member does not overwrite code developed by others. **(VCS)**
- (FR3) Code produced by one team can be integrated with code produced by other teams. **(CI Server + VCS)**
- (FR4) Code is the same during different stages. **(Deploy Manager)**
- (FR5) Different environments serve different purposes. **(Virtualization)**
- (FR6) Different environments are isolated from each other. **(Virtualization)**
- (FR7) A deployment can be easily rolled back if there is a problem. **(Deploy Manager)**



Tooling



PERIODIC TABLE OF DEVOPS TOOLS (V3)

EMBED DOWNLOAD ADD

Os	Open Source	🟪	Source Control Mgmt.	🟩	Deployment	🟦	Analytics
Fr	Free	🟧	Database Automation	⬛	Containers	⬜	Monitoring
Fm	Freemium	🟦	Continuous Integration	🟩	Release Orchestration	🟦	Security
Pd	Paid	🟥	Testing	🟨	Cloud	🟪	Collaboration
En	Enterprise	🟦	Configuration	🟧	AIOps		

1 Os Gl GitLab																	2 En Sp Splunk
3 Fm Gh GitHub	4 En Dt Datical											5 En Xlr XebiaLabs XL Release	6 Fm Aws AWS	7 Pd Az Azure	8 En Gc Google Cloud	9 En Op OpenShift	10 Fm Sl Sumo Logic
11 Os Sv Subversion	12 En Db DBMaestro											13 Os Dk Docker	14 En Ur UrbanCode Release	15 Pd Af Azure Functions	16 Pd Ld Lambda	17 Fm Ic IBM Cloud	18 Os Fd Fluentd
19 En Cw ISPW	20 En Dp Delphix	21 Os Jn Jenkins	22 Fm Cs Codeship	23 Os Fn FitNesse	24 Fr Ju JUnit	25 Fr Ka Karma	26 Os Su SoapUI	27 En Ch Chef	28 Fr Tf Terraform	29 En Xld XebiaLabs XL Deploy	30 En Ud UrbanCode Deploy	31 Os Ku Kubernetes	32 Fm Cc CA CD Director	33 En Pr Plutora Release	34 Pd Al Alibaba Cloud	35 Os Os OpenStack	36 Os Ps Prometheus
37 Os At Artifactory	38 En Rg Redgate	39 Pd Ba Bamboo	40 Fm Vs VSTS	41 Fr Se Selenium	42 Fr Jm JMeter	43 Os Ja Jasmine	44 Pd Sl Sauce Labs	45 Os An Ansible	46 Os Ru Rudder	47 En Oc Octopus Deploy	48 Os Go GoCD	49 Os Ms Mesos	50 Pd Gke GKE	51 Fm Om OpenMake	52 Pd Cp AWS CodePipeline	53 Os Cy Cloud Foundry	54 En It ITRS
55 Os Nx Nexus	56 Os Fw Flyway	57 Os Tr Travis CI	58 Fm Tc TeamCity	59 Os Ga Gatling	60 Fr Tn TestNG	61 Fm Tt Tricentis Tosca	62 Pd Pe Perfecto	63 En Pu Puppet	64 Os Pa Packer	65 Fm Cd AWS CodeDeploy	66 En Ec ElectricCloud	67 Os Ra Rancher	68 Pd Aks AKS	69 Os Rk Rkt	70 Os Sp Spinnaker	71 Pd Ir Iron.io	72 Pd Mg Moogsoft
73 Fm Bb BitBucket	74 En Pf Perforce HelixCore	75 Fm Cr Circle CI	76 Pd Cb AWS CodeBuild	77 Fr Cu Cucumber	78 Os Mc Mocha	79 Os Lo Locust.io	80 En Mf Micro Focus UFT	81 Os Sl Salt	82 Os Ce CFEngine	83 En Eb ElasticBox	84 En Ca CA Automic	85 En De Docker Enterprise	86 Pd Ae AWS ECS	87 Fm Cf Codefresh	88 Os Hm Helm	89 Os Aw Apache OpenWhisk	90 Os Ls Logstash
			91 En Xli XebiaLabs XL Impact	92 Os Ki Kibana	93 Fm Nr New Relic	94 En Dt Dynatrace	95 En Dd Datadog	96 Fm Ad AppDynamics	97 Os El ElasticSearch	98 Os Ni Nagios	99 Os Zb Zabbix	100 En Zn Zenoss	101 En Cx Checkmarx SAST	102 En Sg Signal Sciences	103 En Bd BlackDuck	104 Os Sr SonarQube	105 Os Hv HashiCorp Vault
			106 En Sw ServiceNow	107 Pd Jr Jira	108 Fm Tl Trello	109 Fm Sl Slack	110 Fm St Stride	111 En Cn CollabNet VersionOne	112 En Ry Remedy	113 En Ac Agile Central	114 Pd Og OpsGenie	115 Pd Pd Pagerduty	116 Os Sn Snort	117 Os Tw Tripwire	118 En Ck CyberArk Conjur	119 En Vc Veracode	120 En Ff Fortify SCA

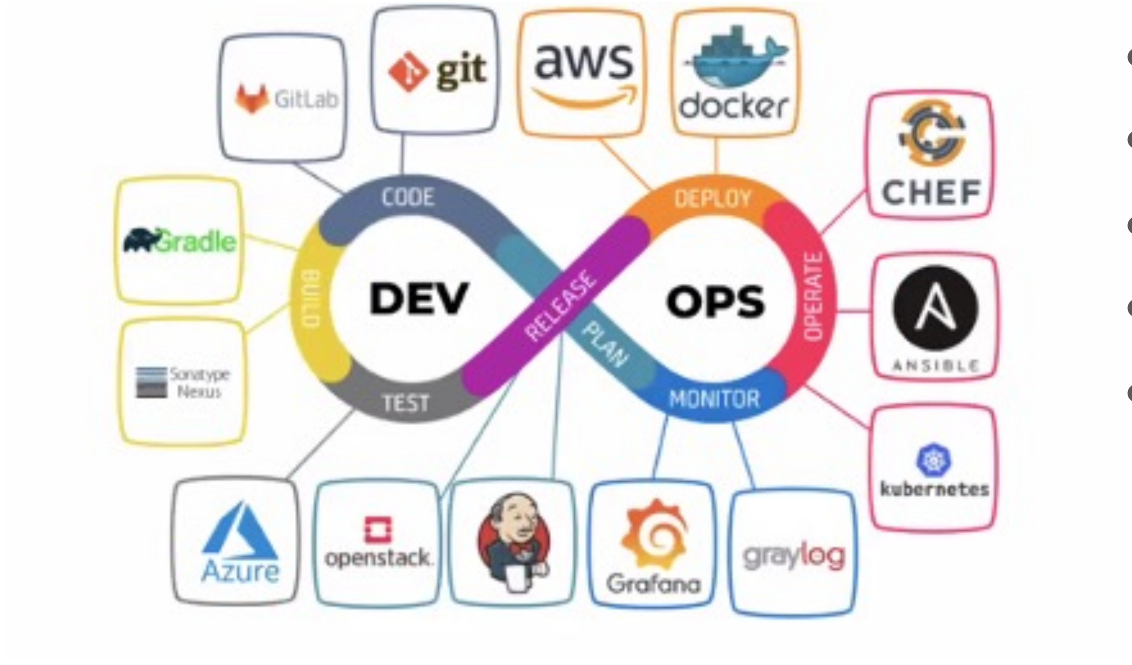
XebiaLabs Enterprise DevOps

Follow @xebialabs

Publication Guidelines
Download

Ref: https://xebialabs.com/periodic-table-of-devops-tools/

Popular DevOps tools



- Version control
- Containerisation
- Orchestration
- Automation
- Collaboration
- Monitoring

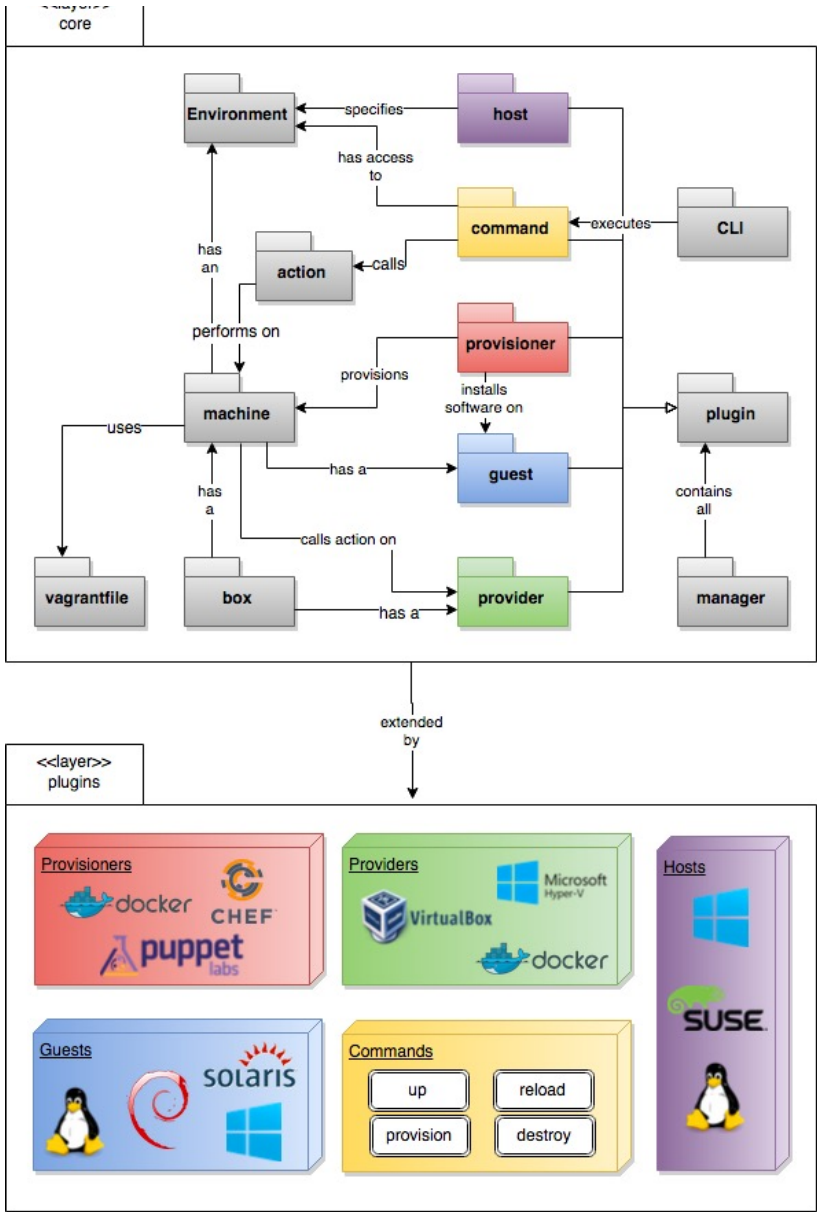
https://medium.com/clarusway/popular-devops-tools-review-ee0cffea14ec

DevOps environment: candidate tools

- Environments setup and configuration
  - Vagrant + Ansible as provisioner to produce a virtual environment to be played with VirtualBox
- PC/Laptop
  - Eclipse (IDE)
  - Java SDK + JUnit + Maven + Tomcat (inside the Dev Env.)
- CI Server
  - Jenkins (CIS and Deploy Manager)
  - GitLab (VCS)
- Testing Server
  - Java SDK + Selenium + Tomcat (inside a Stage Env.)
- Production Server
  - Java JRE + Tomcat

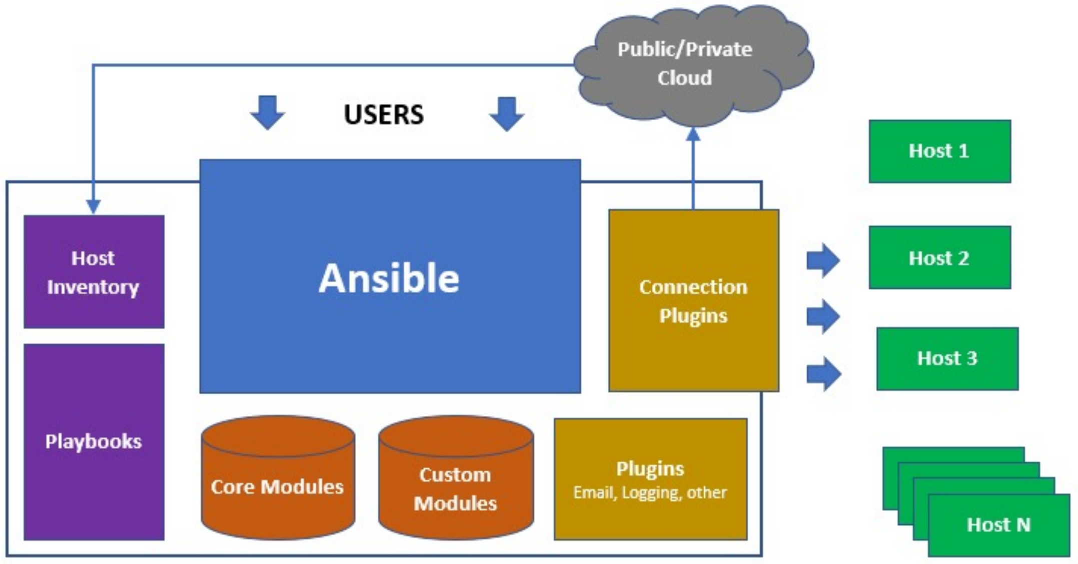
Vagrant

- Vagrant uses "Provisioners" and "Providers" as building blocks to manage the development environments.
- Provisioners are tools that allow users to customize the configuration of virtual environments: Puppet and Chef are the two most widely used provisioners in the Vagrant ecosystem.
- Providers are the services that Vagrant uses to set up and create virtual environments.
- Support for VirtualBox, Hyper-V, and Docker virtualization ships with Vagrant, while VMware and AWS are supported via plugins.

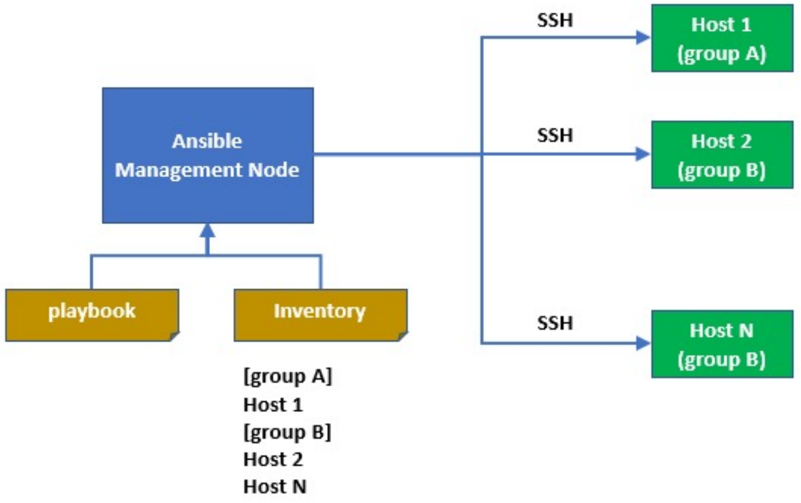


https://delftswa.github.io/chapters/vagrant/

Ansible



Ansible architectural context

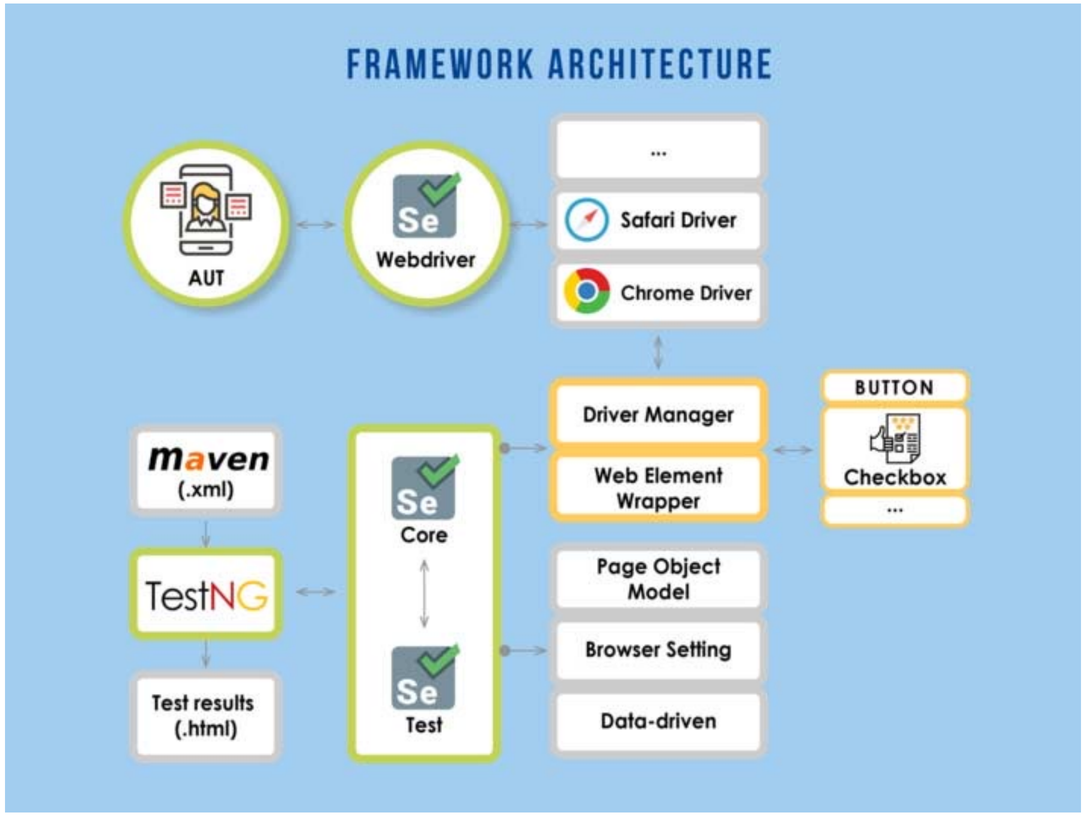


Ansible behavior

https://geekflare.com/ansible-basics/

Selenium

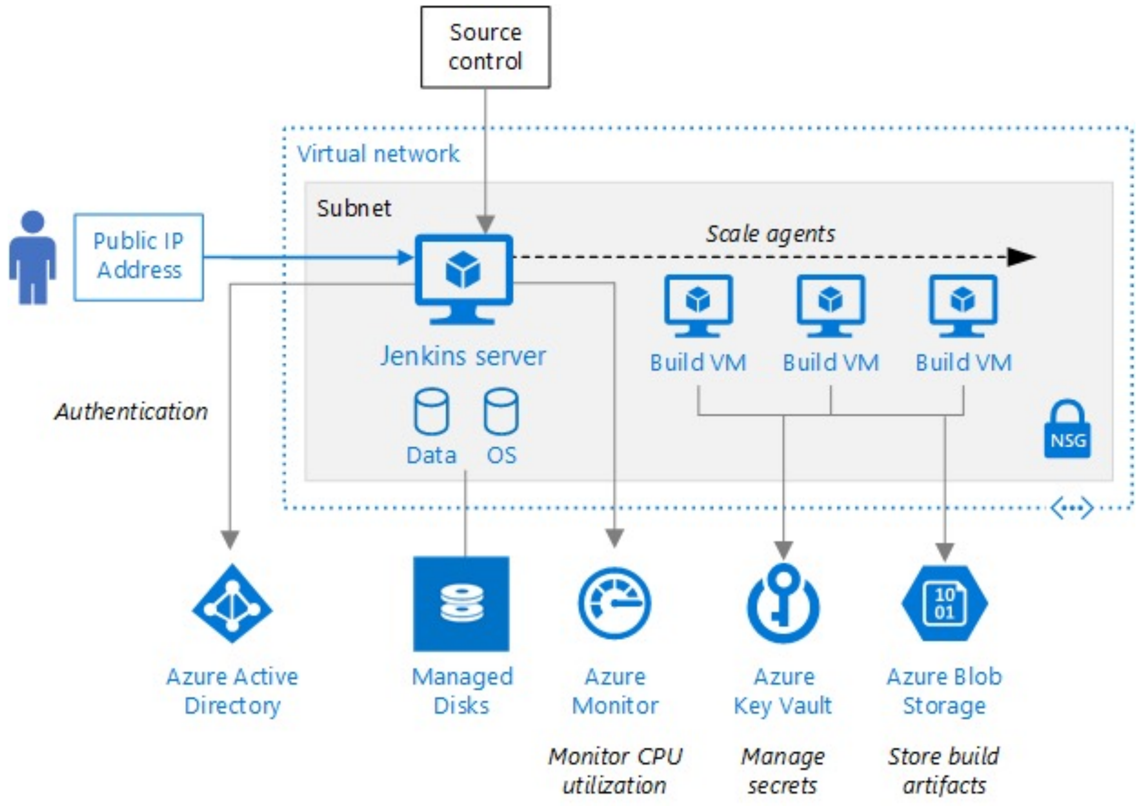
- Selenium is a portable framework for testing web applications. Selenium provides a playback tool for authoring functional tests without the need to learn a test scripting language (Selenium IDE).
- It also provides a test domain-specific language (Selenese) to write tests in a number of popular programming languages, including C#, Groovy, Java, Perl, PHP, Python, Ruby and Scala.
- The tests can then run against most modern web browsers. Selenium runs on Windows, Linux, and macOS. It is open-source software released under the Apache License 2.0.



www.logigear.com/blog/test-automation/building-a-selenium-framework-from-a-to-z/

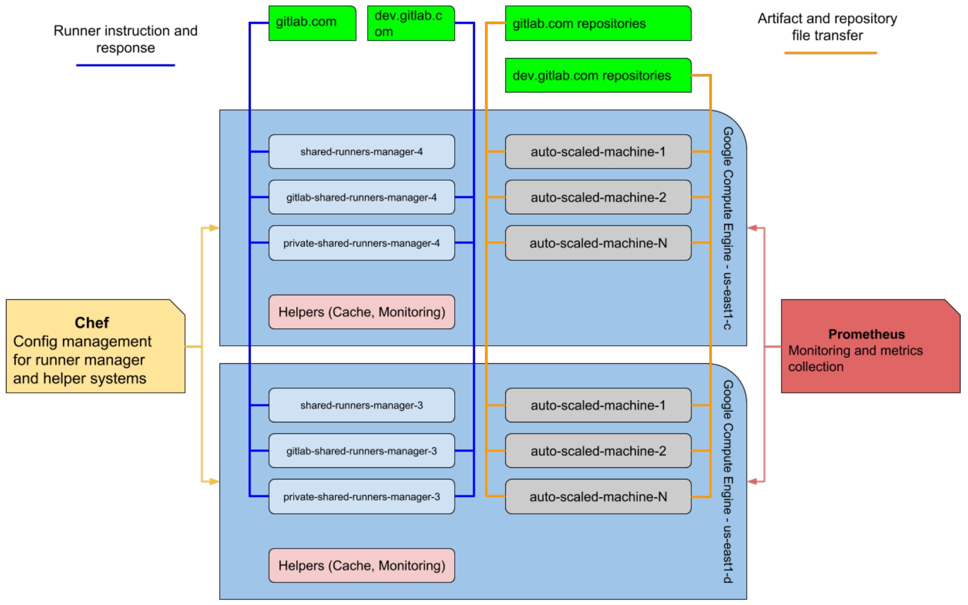
Jenkins

- Jenkins is a free and open source automation server.
- It helps automate the parts of software development related to building, testing, and deploying, facilitating continuous integration and continuous delivery.
- It is a server-based system that runs in servlet containers such as Apache Tomcat.
- It supports version control tools, including AccuRev, CVS, Subversion, Git, Mercurial, Perforce, ClearCase and RTC, and can execute Apache Ant, Apache Maven and sbt based projects as well as arbitrary shell scripts and Windows batch commands.



GitLab

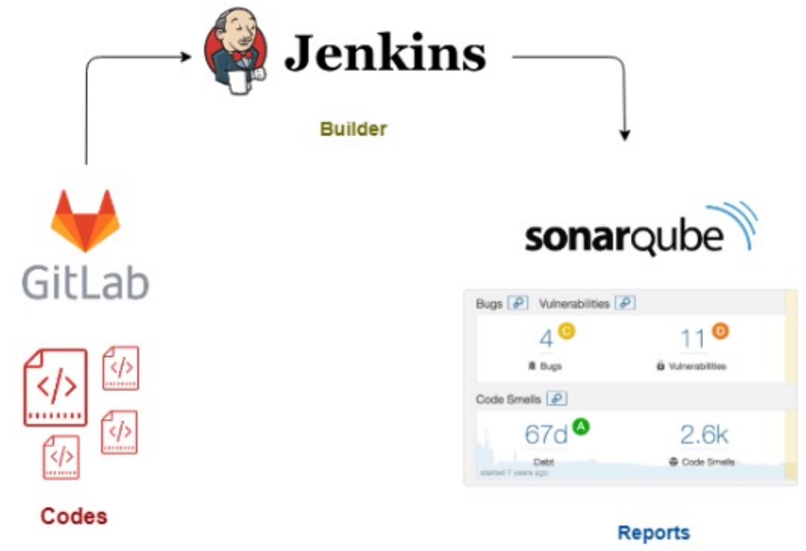
- GitLab is a web-based DevOps lifecycle tool that provides a Git repository manager providing wiki, issue-tracking and continuous integration and deployment pipeline features
- The code was originally written in Ruby, with some parts later rewritten in Go, initially as a source code management solution to collaborate within a team on software development.
- It later evolved to an integrated solution covering the software development life cycle, and then to the whole DevOps life cycle. The current technology stack includes Go, Ruby on Rails, and Vue.js.

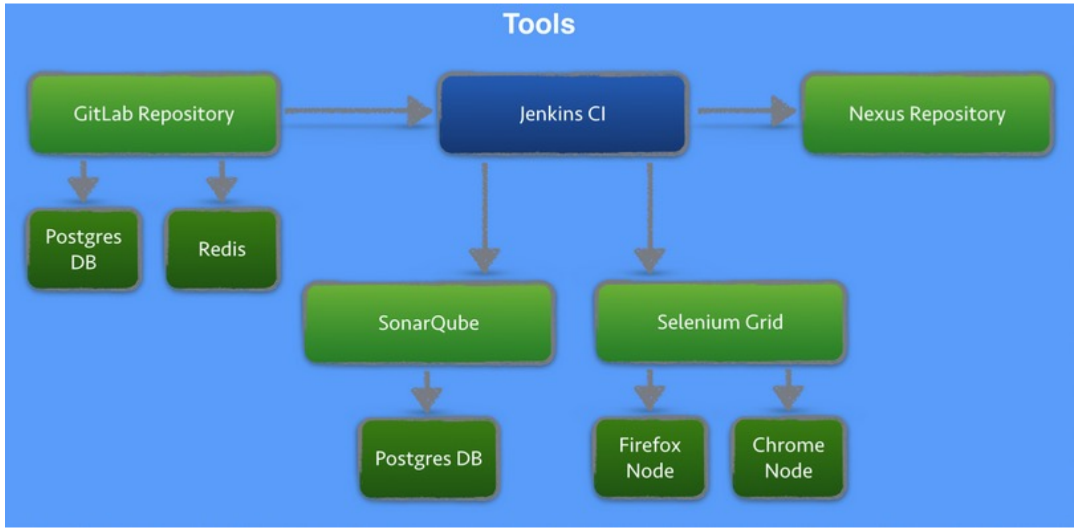


Runner instruction and response
gitlab.com
dev.gitlab.com
gitlab.com repositories
dev.gitlab.com repositories
Artifact and repository file transfer
shared-runners-manager-4
gitlab-shared-runners-manager-4
private-shared-runners-manager-4
Helpers (Cache, Monitoring)
auto-scaled-machine-1
auto-scaled-machine-2
auto-scaled-machine-N
Google Compute Engine - us-east1-c
Chef Config management for runner manager and helper systems
Prometheus Monitoring and metrics collection
shared-runners-manager-3
gitlab-shared-runners-manager-3
private-shared-runners-manager-3
Helpers (Cache, Monitoring)
auto-scaled-machine-1
auto-scaled-machine-2
auto-scaled-machine-N
Google Compute Engine - us-east1-d

https://about.gitlab.com/handbook/engineering/infrastructure/production/architecture/ci-architecture.html

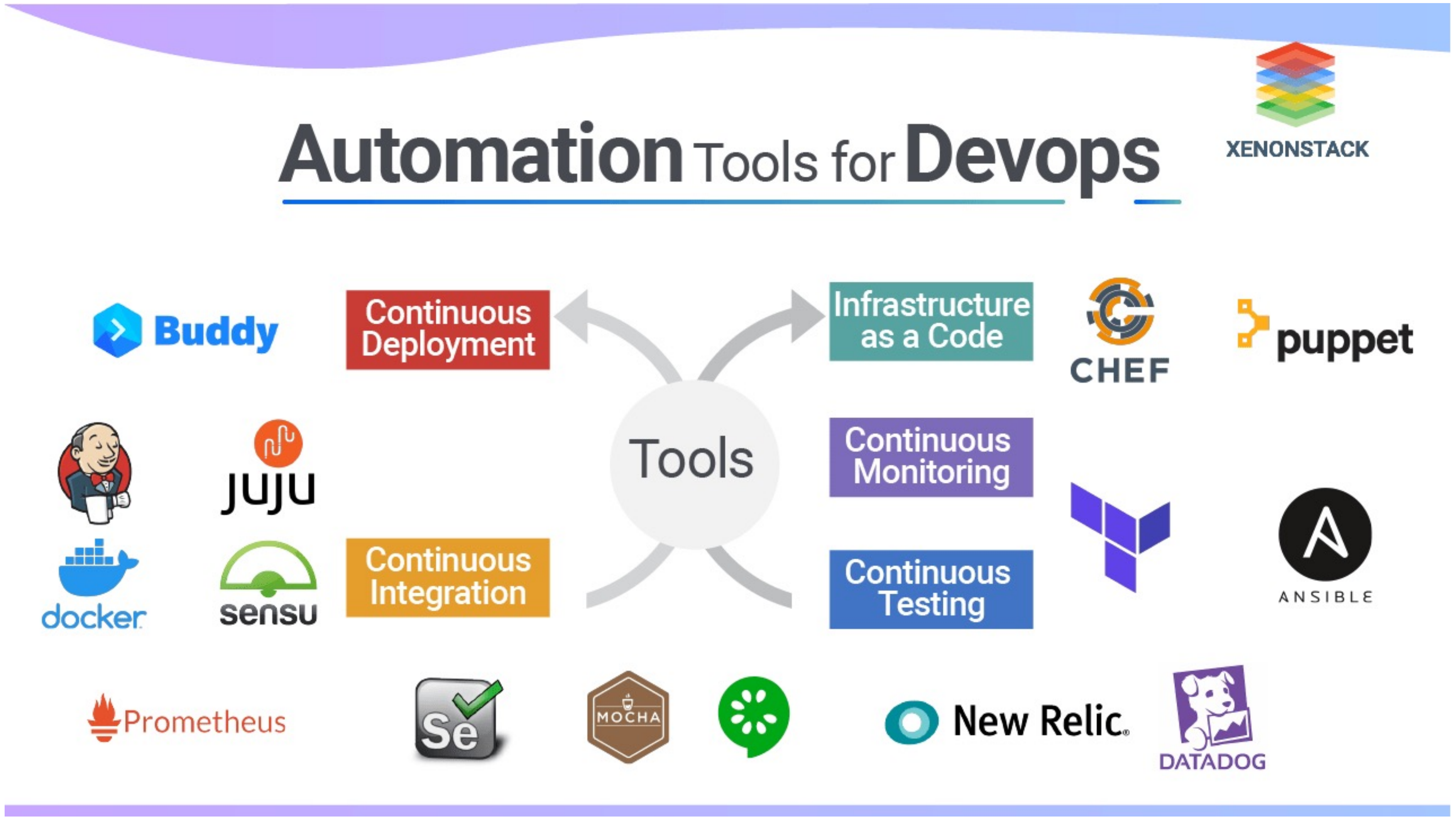
Pipelines based on GitLab (examples)





stackoverflow.com/questions/42564051/is-it-possible-to-integrate-sonarqube-jenkins-and-gitlab-all-in-dockers

linuxhandbook.com/ci-with-gitlab-jenkins-and-sonarqube/



https://www.xenonstack.com/blog/best-automation-tools-for-devops

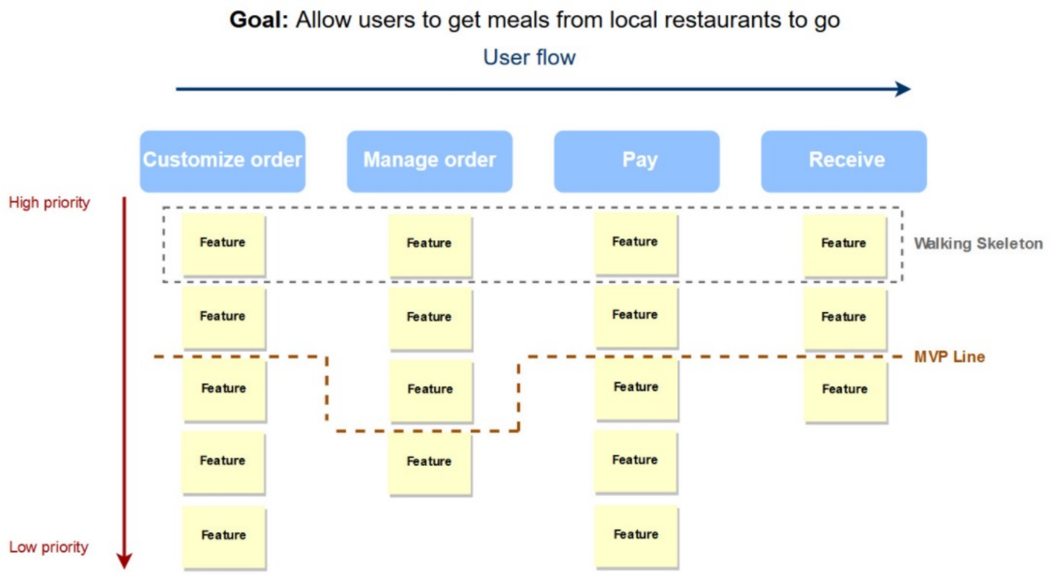
Implementing a Deployment Pipeline

1. Create walking skeleton*
2. Automate build process
3. Automate deployment process
4. Automate Integration Tests
5. Automate Acceptance Tests
6. Evolve

(*) There is an initial step aimed at doing a Value stream mapping. For the sake of simplicity it is skip.

Walking skeleton

A Walking Skeleton is a tiny implementation of the system that performs a small end-to-end function. It need not use the final architecture, but it should link together the main architectural components. The architecture and the functionality can then evolve in parallel.



https://lazaroibanez.com/management-walking-skeleton-the-simple-prioritization-technique-for-mvps-5d99490dad59

Create walking skeleton

- Aim: smallest possible amount of work to get the **key** elements in place.
- Key elements
  - Build process: create the “binaries”
  - Unit test process: create and run a single unit test that asserts “true”
  - Release process: copy the “binaries” where they can be executed (production environment)
    - Run the “binaries” to check that it works
  - Deploy process: copy the “binaries” where they can be executed (staging environment)
  - Integration/Acceptance Test process: create and run single integration/acceptance test against the deployed copy

Note: processes may or may NOT be automated.

Automate build process

1. Create a script to automate the build process
    a. Use a build manager (e.g. maven or gradle)
2. Launched the script every time a developer checks in the VCS
    a. Setup a CIS (e.g. Jenkins) to:
        i. watch the VCS
        ii. check out/update source code
        iii. run the automated build script
        iv. store the binaries where they can be accessed by the team

Automate deployment process

1. Use the environment where you deployed the system for the very first time
2. Create scripts to:
    a. package the system
    b. install the system
    c. configure the system
3. Write and run (automated) deployment tests to verify the system has been successfully deployed
    a. Smoke tests.

Note 2: process should be the same on ANY environment (e.g. stage, production).

Automate Integration Tests

1. Create a script to run test cases
    a. Unit tests
    b. Integration tests
    c. (Some) acceptance tests
        i. Note: the full set should be fast to run (less than 5’)
2. Launch the script after building the system
    a. Launch it into an Integration Environment

Note: you might also include static analysis tools (e.g. SonarQube) in this phase.

Automate Acceptance Tests

Note: you might also include tests to assess NFR.

1. Write script(s) to start up the testing framework(s)
   a. Start it up into the Stage Environment
2. Launched ONLY if smoke tests have passed.
   a. Evidence the system is running
   b. Cover the most important functionality of a component or system
   c. Reveal simple, but severe failures
3. Synonymous
   a. confidence testing
   b. sanity testing
   c. build verification test
   d. build acceptance test

Evolve

- Add stages (as many as required)
- Automate release
  - “Release” <=> “Deployment” into Production Environment

Measure the time (from commit to release). Can you do it better?

$\downarrow$

Recall: trade off between “time-to-market” and “risk that something goes wrong”

Metrics

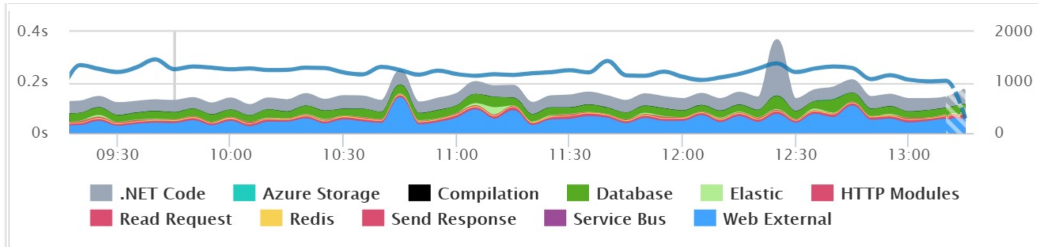
- **Cycle time**: the time that goes since stakeholders decide that a particular new feature has to be implemented until such a feature is delivered to the end-users.

More metrics:

- Lead time: the time that occurs between starting on a work item until it is deployed.
- Velocity: the rate at which your team delivers working, tested, ready for use code.
- Duration of build, including automated tests.
- Automated test coverage.
- Properties of the codebase such as the amount of duplication, style problems, and so on (it can be detected by static analysis).
- Number of defects (bugs found).
- Number of commits to the version control system per day
- Number of builds per day.
- Number of build failures per day.

… and even more about metrics in “15 Metrics for DevOps success”

Metrics



1. Deployment frequency
2. Change volume
3. Deployment time
4. Lead time
5. Customer tickets
6. Automated test pass %
7. Defect escape rate
8. Availability
9. Service level agreements
10. Failed deployments
11. Error rates
12. Application usage and traffic
13. Application performance
14. Mean time to detection (MTTD)
15. Mean time to recovery (MTTR)

https://stackify.com/15-metrics-for-devops-success/#post-14669-_jx3advhf4n5g

References

1. Bass, L., Weber, I.M., Zhu, L.: DevOps : a software architect’s perspective. Addison-Wesley Professional New York (2015)
2. Effective DevOps: Building a Culture of Collaboration, Affinity, and Tooling at Scale, O'Reilly Media; 1 edition (June 24, 2016)
3. Konchenko, S.: Quality assessment of DevOps practices and tools, Ms Thesis, University of Luxembourg, 2018.
4. Erich F., Amrit C, Daneva M. A qualitative study of DevOps usage in practice. J Softw Evol Proc. 2017;29: e1885. https://doi.org/10.1002/smr.1885
5. Hütermann M.: DevOps for Developers, Apress; 1 edition (September 12, 2012).
6. IEEE. 2675 - DevOps - standard for building reliable and secure systems including application build, package and deployment. https://standards.ieee.org/develop/project/2675.html
7. Meyer, B: Agile!: The Good, the Hype and the Ugly, Springer International Publishing (2014).
8. Humble J., Farley D.: Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation (Addison-Wesley Signature Series (Fowler)) Addison-Wesley Professional; 1 edition (August 6, 2010)
9. ISO/IEC: ISO/IEC 25010 - Systems and software engineering - Systems and software Quality Requirements and Evaluation (SQuaRE) - System and software quality models. (2011) ISO/IEC 13211-1.
10. Bass, L., Klein, J.: Deployment and Operations for Software Engineers, ISBN-13: 978-1096269601, Independently published (29 April 2019).

Questions?

Exercise

Recall: Four different times are implied in DevOps definition

*“DevOps is a set of practices intended to reduce the time between **committing** a change to a system and the change being **placed into normal production**, while ensuring high quality.”* [1]

1. **(P1) Pre-commit**: The time during which a developer has sole access to the code.
2. **(P2) Commit-Deployment**: The time during which the code is being tested against other portions of the system and being prepared for deployment.
3. **(P3) Probation**: The time between the code being placed in production (or production-like) and the developers/operators having confidence in its quality.
4. **(P4) Normal production**: The time after the developers/operators have confidence that the quality is equal to other portions of the production system.

TODO: place the times over the process model shown on slide 4.