Introduction module devops

Definitions

- DevOps is a set of practices that combines software development (Dev) and IT operations (Ops)
- The DevOps movement had its beginnings around 2009 when John Allspaw and Paul Hammond from Flickr presented how Dev and Ops can cooperate to deliver 10+ deploys per day
- the ultimate goal of DevOps is to build digital pipelines taking code from the Developer and shipping it to customers in an automated, reliable and secure way.

Definitions

Developers (Devs) build & ship features as quickly as possible while the IT Operations (Ops) deploy code from Devs and make sure systems never go down and fix them if they do so. Before DevOps, there was a conflict of interest because Ops were protecting the stability of systems, and they didn’t welcome changes made by Developers with open arms as they were constantly bringing instability.

Definitions

- Long time ago IT companies used to have servers in their basements and System Administrators (IT Operations) had to configure them by hand. At the beginning it was manageable as you had 1 or 2 servers only, but over time you needed more and more of them and you started to experience troubles.
- Shift from SysAdmin to SRE
- Shift from bare metal to Infrastructure as code (Iaac) and cloud
- Site-reliability engineering (SRE): same as Devops but rather than apply to application layer apply to Infrastructure and ‘shared’ services
- They both share the objective of reducing downtime

From SysAdmin to SRE

Roles in PME to GE

- SYSADMIN (focus on network, security, storage, bare metal)
- SRE (focus on monitoring and alerting, shared components administration, databases, zero downtime of shared components is the objective)
- Devops (focus on test/release/deploy of apps layer, zero downtime and team productivity is the objective)
- Developers / product definition / product owner

Google DevOps manifesto

- 1. Reduce organization silos — It’s about merging your Dev & Ops teams — sometimes it’s as easy as putting them in one room. These two teams should share ownership and use the same tools.
- 2. Accept failure as normal — It’s almost impossible to develop without any bugs at all, and the only way to deal with them is to have a plan of how to resolve the problems e.g. have a script to roll back deployment when it fails.

Google DevOps manifesto

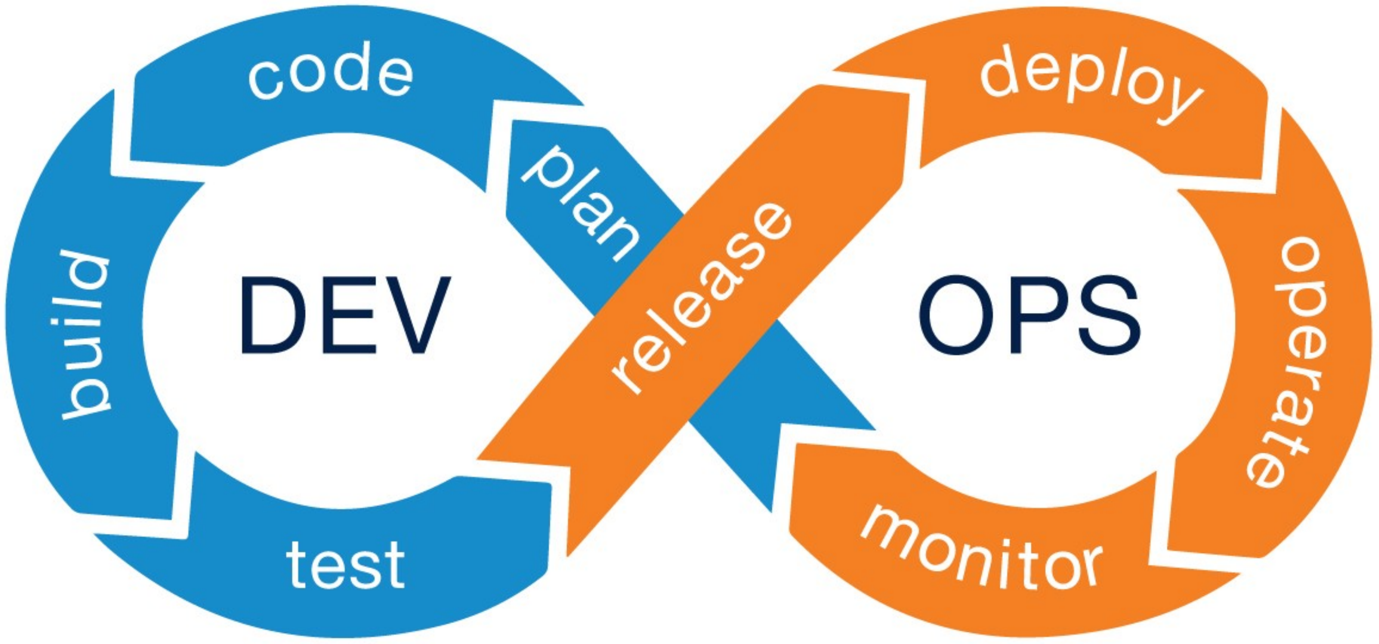
- 3. Implement gradual change — Long time ago some companies used to release large, annual software updates which caused total chaos and disruption. If you deploy 100k lines of code at once finding all bugs would be a miracle. It’s much safer to do it more often in small chunks — preferably in CI/CD methodology.
- 4. Leverage tooling & automation — Ops had a lot of manual work that doesn’t scale — think about installing packages, monitoring, log analysis and so on. In the DevOps world, it has to be automated to increase the velocity of deployments as well as stability & reliability — people are bad at dull, repeatable tasks and often make mistakes.

Google DevOps manifesto

- 5. Measure everything — You always have to have numbers to prove that what you are doing makes sense and it’s worth to continue living the DevOps way. You can do it by introducing measurement, monitoring (e.g. how much CPU you are using), and alerting tools which can tell you or alert you if any of the crucial parts of your services go down.

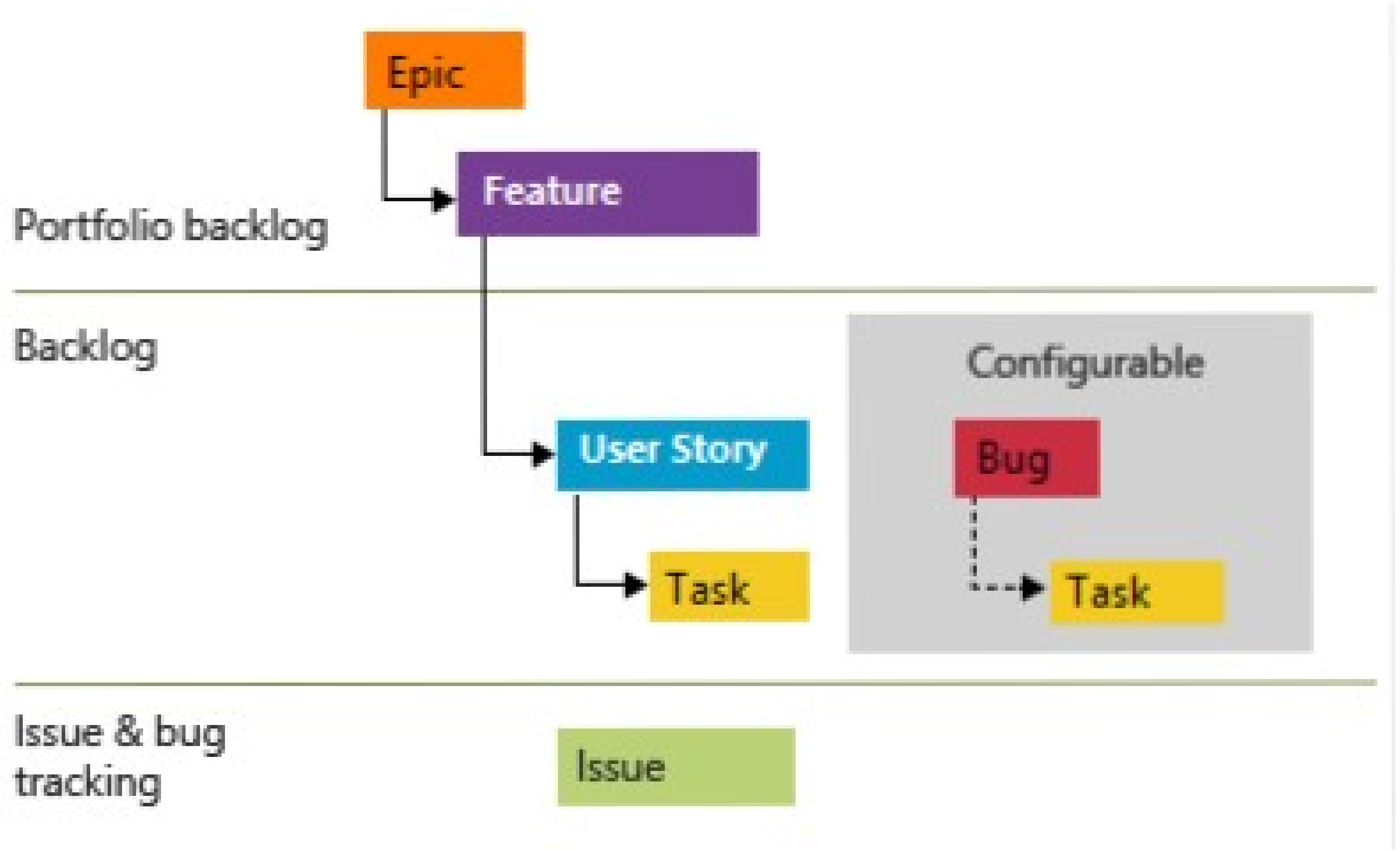
Productivity related topics

- Continuous integration
- Continuous deployment



Productivity related topics

- Agile methodology / short cycle of release

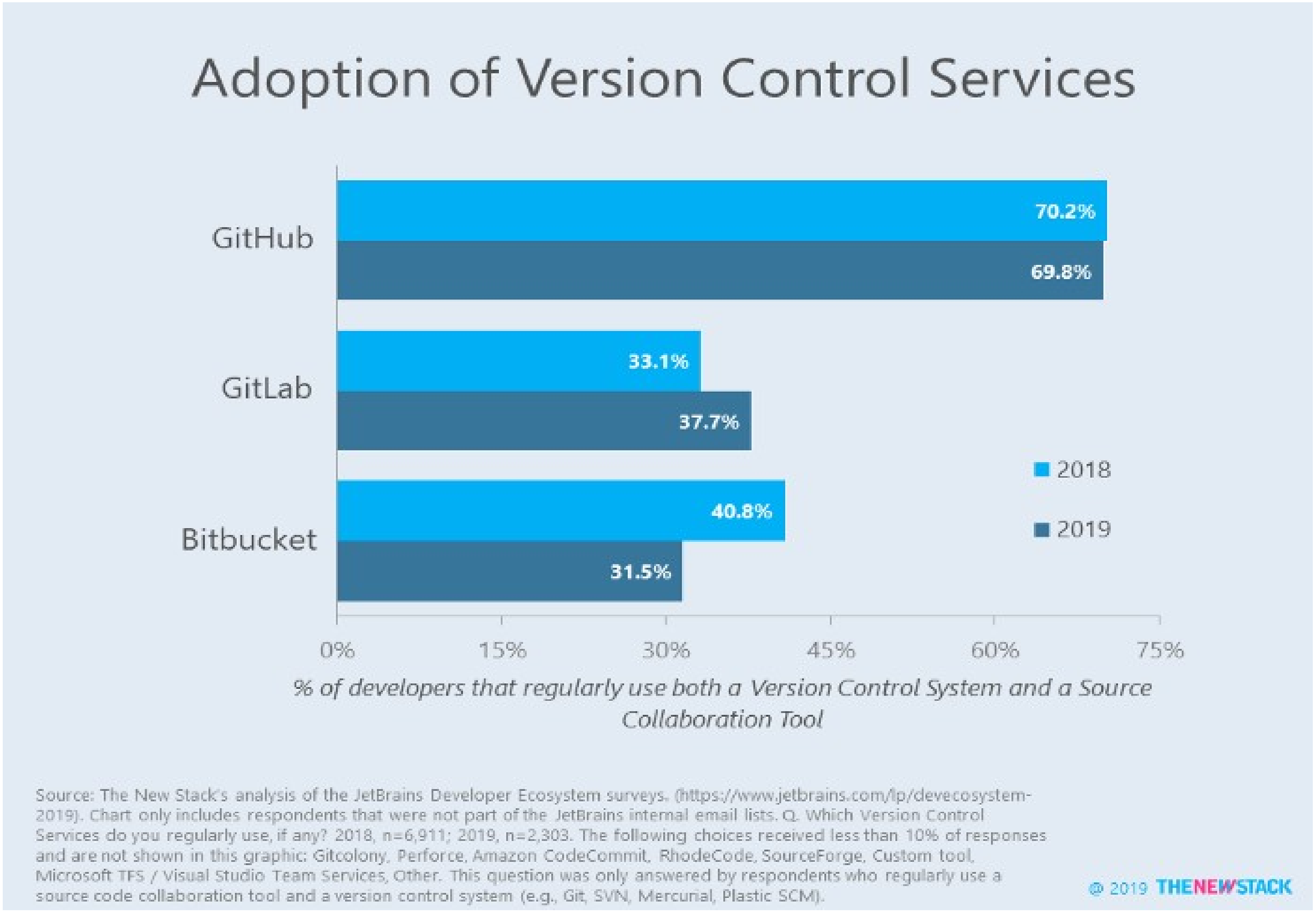


CI/CD

- Integration can be understood as merging code from different Developers into one. More frequent integrations mean that the merge conflicts are easier to resolve, so there is a preference for doing it several times a day — almost continuously. Before the code is integrated into the mutual repository, it should be built (building is converting source code into working software) and tested.

CI/CD tooling

- 1. Version Control Systems (VCS)



Source: The New Stack's analysis of the JetBrains Developer Ecosystem surveys. (https://www.jetbrains.com/lp/devecosystem-2019). Chart only includes respondents that were not part of the JetBrains internal email lists. Q. Which Version Control Services do you regularly use, if any? 2018, n=6,911; 2019, n=2,303. The following choices received less than 10% of responses and are not shown in this graphic: Gitcolony, Perforce, Amazon CodeCommit, RhodeCode, SourceForge, Custom tool, Microsoft TFS / Visual Studio Team Services, Other. This question was only answered by respondents who regularly use a source code collaboration tool and a version control system (e.g., Git, SVN, Mercurial, Plastic SCM).

CI/CD tooling

- 2. Build automation: specific of language used (es. Java – maven, python - pip)

- 3. Artifact repository (es. Nexus / Artifactory)
- 4. Testing: always provide non regression test along the dev process
- 5. Infrastructure Provisioning: tools for launching and manage your Iaac (Terraform, Ansible)

CI/CD tooling

- 6. Containers: docker is the market leader
- 7. Cloud: to avoid having to maintain at all an ifrastructure but be aware this has a cost
- 8. Observability stack: (es. Prometheus + grafana)
- 9. Logging system: (es. ELK stack)