DEVOPS METHODOLOGY IN MODERN SOFTWARE DEVELOPMENT

JAHIC, A.[1] – BUZADIJA, N.[1*]

[1] *Polytechnic Faculty, University of Zenica, Zenica, Bosnia and Herzegovina.*

**Corresponding author*
e-mail: nevzudin.buzadjija[at]unze.ba

(Received 05th January 2023; accepted 21st February 2023)

Abstract. As software engineers, the main idea are to develop systems that will enable progress and make the way of life of other people easier. This paper is a Meta step in solving this problem by giving engineers the tools to improve people's lives through modern software development tools and introducing new concepts that until a few years ago were considered theoretical and have been implemented only in the last few years. It is in practice to replace the current way of software development in the form of eliminating redundant processes in software development and raising new scalability, stability, and efficiency. Through this paper, the author will explain the abstract type of work called "DevOps", where the author will introduce the basic ways of design, implementation, and deployment of software, and explain the main fundamental blocks of this area. Again, the author will go through the concepts of cloud computing, virtualization, containerization, infrastructure as code, and creation of flow structures (pipeline), and the main focus of this paper will be explaining the main point of work of "DevOps" engineers, which is the development of IDP (internal developer platform).
Keywords: *DevOps, IDP, IT operations, CI/CD, virtualization, containerization, deployment*

Introduction

A big misconception regarding software development is that it consists of only development teams who produce code that does not fully integrate with the demands of today's systems. IT Operations present a key to this problem, it consists of highly skilled engineers who are specialized to look at a given system as a single abstracted unit, who think about the requests that the systems fulfill and who create the infrastructure around the given system. Their work allows given software to reach its full potential. They provide networking infrastructure around a given system, implement security mechanisms, produce disaster recovery protocols, and remove redundancy in the developer's work. The solution to the given problem has emerged as a field in software engineering that represents an interface between IT Operations teams and teams of developers, and the name of that field is DevOps.

Discussion

Devops

DevOps at its core represents a collection of rules, conventions in other words "protocols" that are intended to increase the efficiency level of development teams, as well as the IT Operations Teams. Enforcing these rules and standards to both IT Operations and development departments, we ensure the maximum efficiency of both departments. The main focus of the DevOps department is to form a so-called IDP (internal development platform) which will serve as the optimal development standard for both teams. DevOps as a newly formed department consists of 5 main postulates

(Kim et al., 2014): (1) collaborations; (2) automatization; (3) continuous development; (4) resolving client needs; and (5) abstraction (looking at bigger picture). All of these postulates are implemented when developing IDP software which represents an interface between Development an IT Operation teams, it enables the synchronization of the given teams with the guidance of a project manager and results in software that is at its most optimal level and has the maximum efficiency level for the client usage.

Currently, about 90% of infrastructures are based on some type of cloud service. When talking about cloud computing the standard definition is the one provided by the National Institute of Standards and Technology (NIST): "cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction (Mell and Grance, 2011)." Characteristics of every Cloud Provider (Buyya et al., 2013): (1) On-demand self-service: Every cloud provider needs to enable its users to manipulate the resources he or she owns without the involvement of the given cloud provider. The reason for this is to enable the agility that is necessary for the client to adapt its resources to the commercial requests that he or she tries to fulfill; (2) Broad network access: All services and resources must be up and running at all times and accessible via the internet; (3) Resource pooling: All types of upgrades and configurations of the given resources must be available to the end user at any time of the day; as well as (4) Rapid elasticity and Measured service: It needs to be able to calculate the precise metrics and present them to the end user at any time so that the user can successfully allocate necessary resources that he or she needs.

Virtualization

Virtualization is one of the consequences of cloud computing expansion. In an ideal world, it allows the efficient usage of computer resources. Every computer has its main performance parameters (CPU, Memory, Storage, OS), and the main concept behind virtualization is taking physical parameters and partitioning them into smaller segments that are used as independent physical resources inside the main physical resource. For example, the users have a machine with (4CPUs, 8GB RAM, and 1TB SSD), using that technique the users can make 4 “virtually” independent smaller machines each with (1CPU, 2GB RAM, and 256GB SSD), these 4 machines share a common physical resource form which they are provisioned but they represent an isolated system which can be used in work. As in *Figure 1*, there are a large number of benefits when using virtualization: smaller expenses, bigger efficiency, speed of configuration, and ease of work.

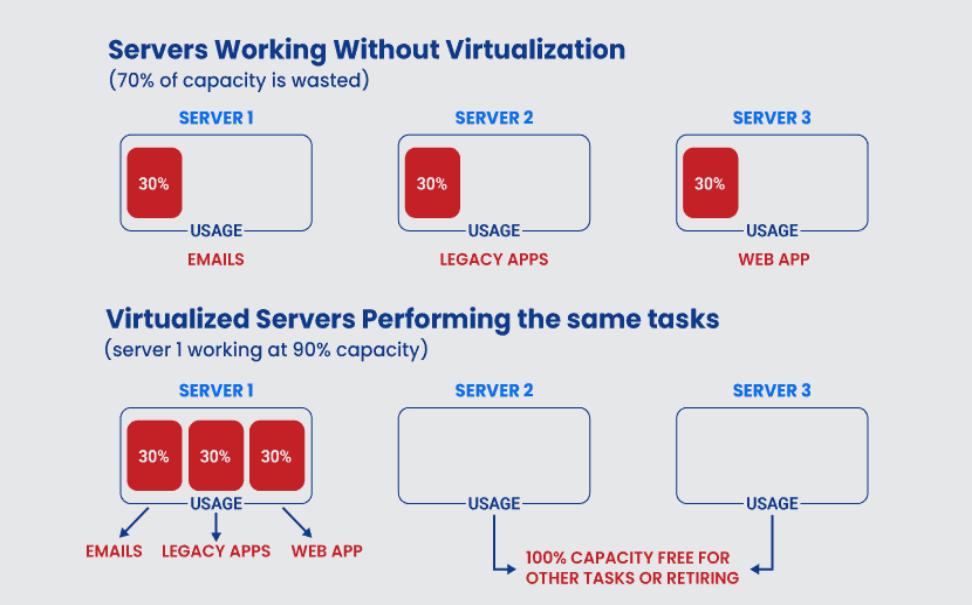

Figure 1. *Graphical representation of the given technique.*
Source: (Rogier and David, 2007)

Every virtual machine can be viewed as an independent system in relations to other systems. Referring to the example of this concept in software testing where the users can make 3 separate test environments (VMs), and each one will test the functionality of the software on a different OS. One virtual machine can run the testing of the app on Linux, the second one can perform the tests on windows and the third one can perform tests on macOS. In practice, the author divides 2 types of virtual machines that can implement on the systems: Hypervisor Type 1 and Hypervisor Type 2. Hypervisor is a software layer between the virtual machines and hardware from witch is being provisioned from. It enables that every virtual machine gets enough resources that needs for later work. Using Hypervisors, the author will use to manipulate the virtual machines. Hypervisors have the role as a scheduler in a way that they don't allow already provisioned resources for one virtual machine to be reprovisioned to another. It keeps the virtual machines on the same network and decides about their visibility. Host OS can provision a finite amount of Guest OS which can be used later, and that process is usually done via a graphical interface for virtualization.

IaC (Infrastructure as Code)

The way of providing efficient DevOps solutions used by modern engineers is by using a specific functionality from a larger set of tools or apps that can provide us with a given result. All manifests, scripts, and configuration files are almost always kept in git repositories; they are being controlled via git systems on machines which is at its core interpreted as a program in the operating system of the machine. Transitive property deduction gives the conclusion that all of the manifests, scripts, and configuration files are nothing more than instructions loaded via the command line as some kind of code. The first tools for configuration management are ones like Chef, Puppet, and Ansible (*Figure 2*). They are the ones who for the first time combined the declarative commands the author use in programming with a problem we are trying to solve and that is the transition from Desired State to Current State. They allow us to use serialization

languages like YAML to write templates for configurations, the given template would configure the machine from scratch as it would have been done manually by an engineer beforehand.

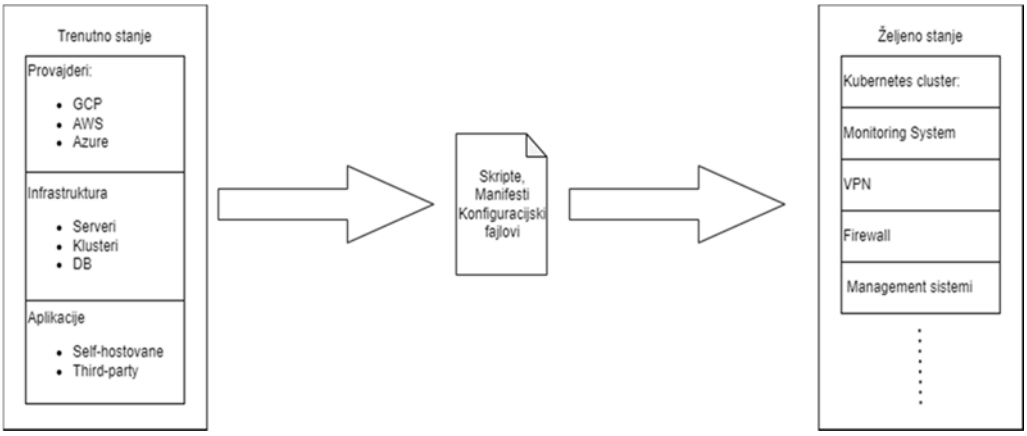

Figure 2. *Schematic representation of state transfer.*

The first tools for configuration management are ones like Chef, Puppet, and Ansible. They are the ones who for the first time combined the declarative commands we use in programming with a problem we are trying to solve and that is the transition from Desired State to Current State. They allow us to use serialization languages like YAML to write templates for configurations, the given template would configure the machine from scratch as it would have been done manually by an engineer beforehand. An example of these kinds of scripts is stated below (Taylor, 2013):

hosts: group2 tasks:
name: sshd config file modify port lineinfile:
path: /etc/ssh/sshd_config regexp: 'Port 28675'
line: '#Port 22' notify:
restart sshd handlers
name: restart sshd service: sshd
name: sshd
state: restarted

For complete usage of IaC, we need software that is in condition to solve the problem of uniformity and immutability, to remove all the repetitive and redundant work that is time-consuming for an engineer. The only tool on the market made universally to solve this issue is Terraform. In *Figure 3*, it clearly stated the process and the role of a tool like terraform in configuration management. The main role is to translate our infrastructure from the Desired state to the Current State.

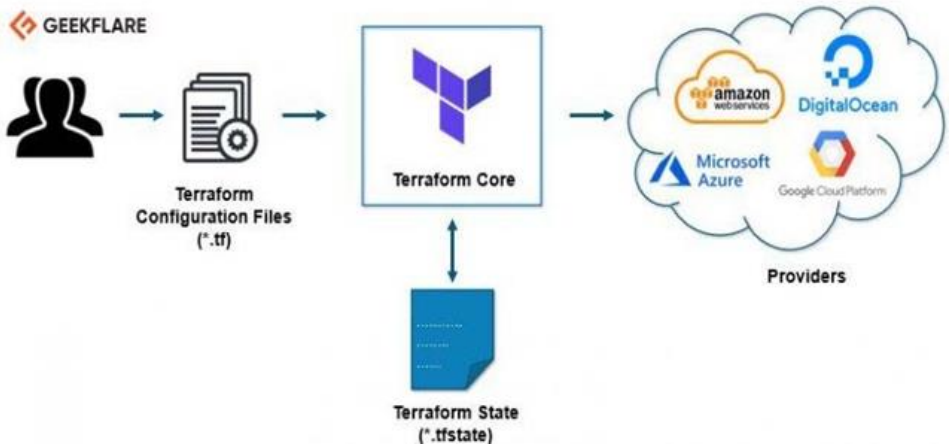

Figure 3. *Translation between the Desired state and Current state using terraform. Source: (Krief, 2019).*

For a better classification of the tools for configuration management, the author will define 4 key segments that are needed in configuration management: initial provisioning of infrastructure; managing current infrastructure; initial application configuration; and managing configured applications (*Figure 4*). There are the basic requirements that tools for configuration management need to solve. Unfortunately, there is no software this time to implement all 4 features. IaC tools like Terraform or Crossplane are Cloud Agnostic. This means that they are compatible with all cloud providers out there. One code can provision from all could providers. This concept increases the complexity but gives us much greater power in resource control. The main idea of these tools is that there is a "core" that performs the defined translation from Desired to Current State in a way that it performs a manifest transformation in these tools and from the given manifest it performs a series of API calls to the needed provider who responds in a way that is defined in the manifest itself.

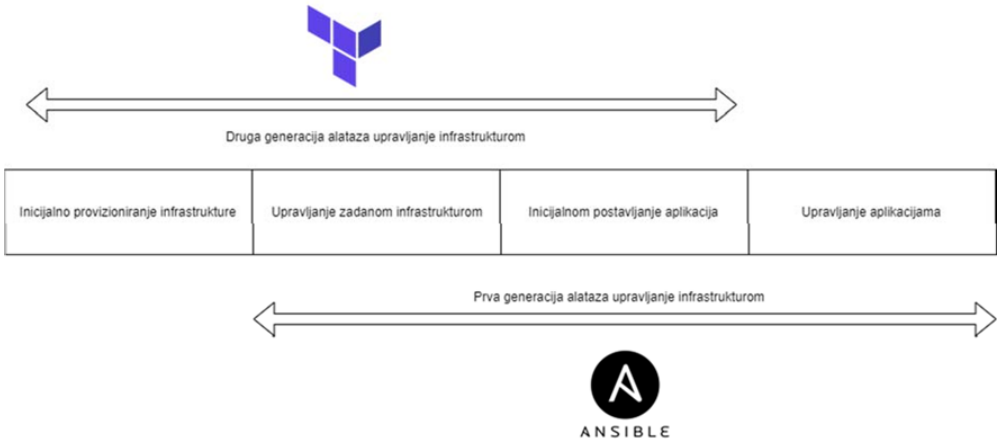

Figure 4. *Segments of configuration management tools.*

CI/CD (Continuous Integration and Continuous Deployment)

The process of deploying given software is at its core very simple. The application gets compiled and the compiler produces the resulting artifacts like (JAR, DLL, WAR). Those files then get copied to the server and replaced with the version of the application

that is currently running. Unfortunately, the issues begin when we fix the small mistakes from the integration process and deployment process on the server itself. The result is that the given server mutates over time. The server gets filled with all sorts of patches, configuration files, and scripts. The bigger issue is in situations that are composed of different environments like (UAT, STG, and LIVE) we cannot be sure of our code functionality because it's not in a "clean environment". As a result, the author gets 2 same applications running on 2 servers with different behavior. This process gives us a small peak into the complexity of successfully performing the process of Continuous Integration and Continuous Deployment. One of the first solutions was forming the VMs for every instance of the given application. This solves the issue of collecting unnecessary scripts, configurations, and manifests from different versions of applications. The process consisted of provisioning a virtual machine on which we install necessary programs and on that machine, we deploy our application. The problem with this is that in a very short time we can reach the chaos mentioned above because it's necessary from time to time to do some corrections on the servers. In a very short time, the author can reach the state in which we have the same application having multiple behaviors on multiple servers but at a much decreased rate because of the more frequent use of virtual machines.

The current solution is to form a server that handles automatic deployment. These types of software often use more than one type of tool and concept. The result is a CI/CD Pipeline that our program needs to pass in order to be active on the server so that the users can use it. To better understand a complex process like this one we need to introduce some basic terminology and classification. Basic classification goes by: continuous integration; continuous delivery; and continuous deployment (*Figure 5*).

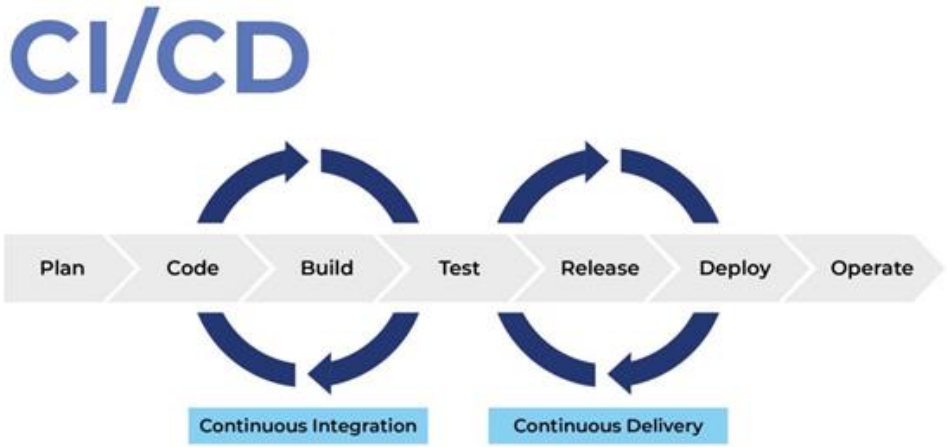

Figure 5. *Graphical scheme of CI/CD.*
Source: (Farcic, 2016).

CI (Continuous Integration)

One of the biggest mistakes when learning CI/CD is the necessary understanding of the first part that relates to Continuous Integration. If there is no Continuous Integration there can no exist Continuous Deployment. This is completely logical because we at least need to provide a system that verifies the integrity and functionality of our code and flags what specific version of the code to deploy. Continuous Integration (CI) defines the starting point of our CI/CD pipeline but does not tell us where it ends. The reason for that is that different companies have different business needs which result in

different levels of integration. By definition, it encompasses Continuous Integration, compiling, and testing inside a given environment. For a CI system to run correctly it's not enough to own a repository where the developers will commit the code and hope that it will work when it comes to the deployment and delivery of the code. It is necessary at least to do a static analysis of the code and, perform a pre-deployment test, compile the application and do the post-deployment test.

Continuous Integration has no a given ending point. Different companies that implement Continuous integration will have very different integration systems; the logical reasoning is that not all programs are the same and should not be treated as the same. This is the reason why is this process very hard to implement in practice, But when it's successfully implemented it gives an enormous boost to the efficiency level (Marijan et al., 2018). The basic steps every Continuous Integration needs to implement: able to commit the code to the centralized repository; statical analysis; pre-deployment test; code compiling; and post-deployment test. A useful way of thinking when designing a Continuous Integration system, it is to make it work as an event-based system. In programming, we have the concept of event-listeners which listen to the changes made by the users when activating so-called triggers that execute a specific function on a given system. Tools that perform the Continuous Integration process are called event listeners that are triggered by the developer when committing a code (*Figure 6*).

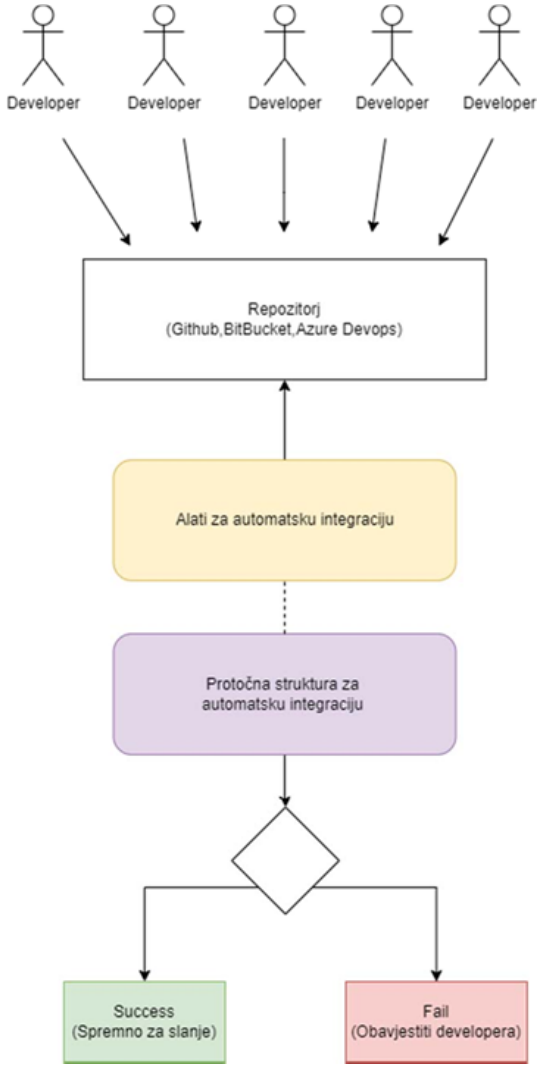

Figure 6. *Process of Continuous Integration.*

The statical analysis presents the first part of the realization of the Continuous Integration Pipeline. It represents syntax checking without compiling the code, and redundancy checking. Tools used for this kind of process are JSHint and PMD. Testing the code before compiling is often one of the most critical parts of a pipeline. This is a mandatory part because it exponentially increases are code functionality. Tests that require this do not require deploying the code to some environment. Unit tests are in this category whilst Integration tests aren't. This is the first obstacle our software needs to pass in order to interact with other components of the system. Compiling the code is the part that varies from a programming language to a programming language. Every language has its own way of compiling its code. In Javascript, we just minimize the file whilst in Java, we compile it into a JAR file that we send to the production. Testing after compiling the code is often the final phase of the pipeline and it's often performed on a temporary environment in which the tester executes the other half of the tests that are necessary for the application. Integration tests are part of this section because the tester needs to make sure that the entire app works and that individual component mutually react to one another with or without errors.

IDP (Internal Development Platform)

Services that developers are integrated into one system are called IDP (Internal Developer Platform) and it's the heart of DevOps. The way that IDPs are designed is characterized in a way that we need to be able to synchronize the entire Desired State with the Current state in our infrastructure indifferent regarding to the internal structure of these states. We need to construct a bridge between these 2 points. The Bridge should be able to act as an interface for every desired Desired state disregarding the cloud computing model being implemented (private, public, hybrid). It should be able to integrate services that are in the Current State that are from third-party vendors and also needs to support all the tools we need to describe our Desired state.

Solution

In today's environments, all the above problems regarding the Deployment of specific software are being transcended by using a universal tool in the form of an internal development platform. Currently, on the market, there is only one candidate for building IDPs and it currently holds the monopoly in designing the IDPs and that is Kubernetes KUBEAPI. This is currently the only universal API that can integrate all the above tools and bring them to one system (*Figure 7*). It is not enough to throw links to a bunch of different services and expect the developer to use them. IDP is not a set of links that we send to the developers as a bag of everything so that they can choose what suits them the best. It is necessary to eliminate the complexity that IT operations possess so that they can just "click" what they want and get the desired result. In contrast, developers should need to know how to use every single service implemented in IDP, which is almost impossible because technology develops too fast. The main takeaway from this system is that EVERYONE should be able to describe the desired state of the infrastructure. Every developer needs to be in an independent state and to have enough knowledge to the provision by himself the necessary resources that he needs without falling behind the team of IT Operations. The process of usage consists of the DevOps engineer writing a template for every possible type of resource that can possibly be provisioned. These are the mentioned CRDs. They present a well-defined resource that is provisioned. For example, 1 backend instance can consist of an application, a database, and an API, for every type of these resources we need to provide a container that has to be on the same network with all the tools that come with them. The developer uses the CRDs to form its own CR (custom resource) and parametrize the given resource in a way to choose the strength of the machine, and database size. It's important to point out that it has full control of the given resource (*Figure 8*).

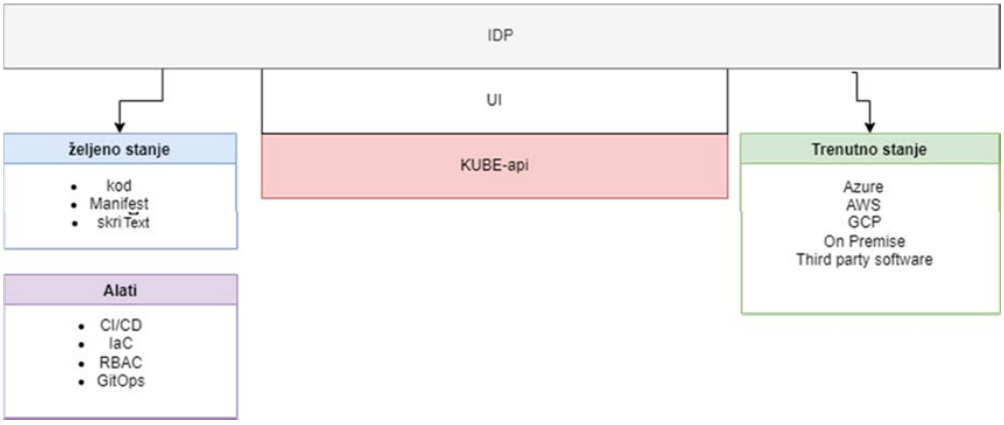

Figure 7. *Graphical schema of the K8 API.*

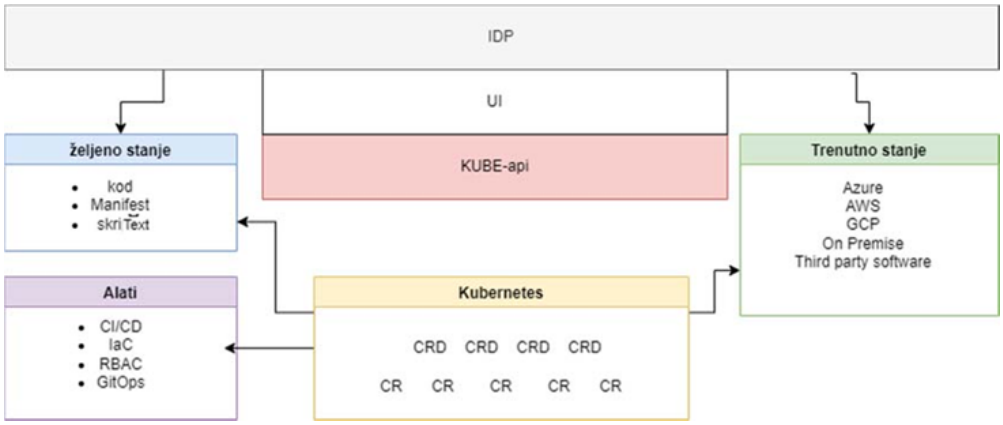

Figure 8. *Schema of an IDP-a.*

Conclusion

DevOps is still a very young department in the area of Information Technologies which means that is subject to a lot of change, concepts that are described here are the first step in differentiating between good and great engineers. The needs of the market are constantly upgrading, applications get more complex, and codes are getting difficult to maintain. IT Operation teams can not manage to fulfill all the necessary requests that are coming. The development of IDPs is a brand new concept in the IT world, and it's also an ideal that DevOps strives to achieve. It's still taken with a dosage of insecurity, but it definitely represents a new way of developing software using the modern methodologies that the field of DevOps is providing to us.

Acknowledgement

The research study is self-funded.

Conflict of interest

The authors declare that the research was conducted in absence of any conflict of interest.

REFERENCES

[1] Buyya, R., Vecchiola, C., Selvi, S.T. (2013): Mastering cloud computing: foundations and applications programming. – Morgan Kaufmann 468p.
[2] Dittner, R., Rule, D. (2011): The Best Damn Server Virtualization Book Period: Including Vmware, Xen, and Microsoft Virtual Server. – Syngress 960.
[3] Farcic, V. (2016): The DevOps 2.0 toolkit: Automating the continuous deployment pipeline with containerized microservices. – CreateSpace Independent Publishing Platform 414p.
[4] Kim, G., Behr, K., Spafford, K. (2014): The phoenix project: A novel about IT, DevOps, and helping your business win. – IT Revolution Press 345p.
[5] Krief, M. (2019): Learning DevOps: The Complete Guide to Accelerate Collaboration with Jenkins, Kubernetes, Terraform and Azure DevOps. – Packt Publishing Ltd. 504p.

[6] Marijan, D., Liaaen, M., Sen, S. (2018): DevOps improvements for reduced cycle times with integrated test optimizations for continuous integration. – In 2018 IEEE 42nd annual computer software and applications conference (COMPSAC), IEEE 1: 22-27.
[7] Mell, P., Grance, T. (2011): The NIST definition of cloud computing. – National Institute of Standards and Technology 7p.
[8] Taylor, D. (2013): Ansible tutorial for beginners: Playbook, commands & example. – Guru99 Official Portal. Retrieved from: https://www.guru99.com/ansible-tutorial.html