Alexander Zaytsev

Continuous integration for kubernetes based platform solution

Helsinki Metropolia University of Applied Sciences
Bachelor of Engineering
Information Technology
Bachelor's Thesis
June 4, 2018

Metropolia

Abstract

Author	Alexander Zaytsev
Title	Continuous integration for kubernetes based platform solution
Number of Pages	37 pages
Date	4 June 2018
Degree	Bachelor of Engineering
Degree Programme	Information Technology
Professional Major	Software Engineering
Instructors	Santeri Anttalainen , Product Owner Dr. Tero Nurminen, Principal Lecturer

There is a high interest in cloud computing among software companies nowadays. There are multiple advantages to move backend services to the cloud or to utilize software through the internet. For instance one does not need to care about maintaining infrastructure. Another trend technology that is going to be described in this thesis is kubernetes. It enables orchestration of container based applications.

An essential part of every software project is testing. In order to achieve frequent releases one need to automate this process. This could be enabled using continuous integration and continuous deployment practices that will be described in the following chapters.

The goal of this project is to develop an automated continuous integration pipeline for for kubernetes based platform solution.Various tools and technologies will be assessed and afterwards used for the implementation: Openstack , Heat , Jenkins to name a few. Moreover some future improvements will be proposed.

Keywords	Microservices,Kubernetes,Continuous Delivery, Continuous Integration, Openstack, Heat, Ansible, Cloud Computing

Metropolia

Contents

List of Abbreviations

CD	Continuous Delivery
CI	Continuous Integration
API	Application Programming Interface
IP	Internet Protocol
YAML	YAML Ain’t Markup Language
SaaS	Software-as-a-Service
PaaS	Platform-as-a-Service
IaaS	Infrastructure-as-a-Service
SaaS	Software-as-a-Service

1 Introduction

Cloud computing has gained immense popularity among software companies. According to Forrester Research global public cloud market will be $178 billion in 2018 [1]. There are already big players on the market : Amazon Web Services, Microsoft azure and Google Cloud. Nevertheless middle size companies are also coming to the market. All this interest is due to the benefits that cloud computing brings. For instance backend services in cloud allows companies not to build their own infrastructure, moreover security and backups are also handled by cloud providers. Moreover, kubernetes extensible open-source platform is widely spread nowadays. It provides possibilities for creating own platform solution that can conduct containerized applications.

Development of an application consists of different stages: designing, coding, testing,building ,releasing ect. The main goal is to satisfy customer needs. In order to achieve this companies should do releases as often as possible. In this case new functionalities or bug fixes will be deployed faster. Continuous integration practises help to accomplish these requirements.

The goal of this thesis is to study the best practices of continuous integration and to come up with an automation pipeline. The key value is automation of processes in all stages and a solution that does not stack the development . There are five more chapters in this thesis.

In the second chapter one can find study for the theoretical background of this project. There will be a literature review and key concepts that describe the topic relevant for this work.

After that in the third chapter, that will consist of description of the project and tools that will be used to implement the automated testing pipeline.

Section number four is the main part of the thesis. The implementation will be presented in this chapter. There are multiple steps that should be addressed to accomplish it.

In the next chapter one can find evaluation of results that were achieved in the previous section. Moreover the discussion of future development enhancements for this thesis will be proposed.

The last but not least chapter will hold the summary and some general thoughts about achievements in this project.

2 Theoretical background

2.1 Microservices

The market of applications is constantly growing. End user expects to receive quality service with rich functionality,modern and perceivable design and availability 24/7. In order to satisfy these demands companies need to roll out updates as frequently as possible. It used to be a common practise to develop software as a large, monolithic application. Typical monolithic example has a complex system design and all the business logic running in one module. Developer would need a solid understanding of the whole code base in order to commit to it. Moreover any change should go through an extensive number of regression tests, which will heavily increase the development cycle . [3]

Microservice architecture implies to have small components that are not coupled together. Each module is doing its own task and independent from other blocks. It is beneficial to have small components, because it would lead to faster testing and deployment. REST APIs are used for communication between the services.

2.1.1 Characteristics of Microservices

There are several characteristics that needs to be reviewed in order to have a better understanding of microservice architecture style.

Codebase

The code size of particular microsirvice component is relatively small and its functionality should be limited to accomplish particular task. In this case it would be easier for new developer in the team to read and contribute. Moreover building and running application for development purposes is much faster with smaller blocks. Another advantage to be mentioned is the possibility to choose the the most suitable

programming language, library or framework for each service. It is easier to adopt or try out new technologies in system with independent modules. [4]

Deployment

In monolithic approach even a small change would require building the whole project. This could lead to the situation in which application has rare releases and with massive changes. In microservice development it is encouraged to make faster and more frequent releases. When a service is deployed independently is it easier to trace failures and if problem occurs it would not take much time to roll back to previous version.

Scalability

In order to scale a large application a duplication of the whole project is needed. Thus scaling such project would require abundant resources. Moreover what if some functionality would need even more resources. It is becoming obvious that copying large systems is not efficient. In contrast each microservice could have different number of copies, which would increase the performance of the application and enhance user experience. [4]

Composability

Modern applications needs to be developed for different use cases. Web, native phone apps, tablet , wearables to name a few. Microservices are independent and reusable components and they could be operated for various platforms. Moreover code isolation means that a developer does not need to bother about other component implementation. And if there is a legacy code that needs to be refactored it could be done in parallel or when the appropriate time comes. Thus team distribution is becoming more efficient process .

2.1.2 Challenges of Microservices

Microservice architecture has many advantages in comparison with monolithic. Ease of deployment, scaling , healing to name a few. Nonetheless there are several challenges that come along with microservice architecture.

Complexity. Each microservice component is an independent module, which works in a bundle with other isolated elements. Therefore handling communication between services becomes a complicated task. For instance there would be a need to provide API endpoints for every module. Numerous number of APIs requires well written documentation.

Testing. In comparison with monolithic architecture it is easier to test small, independent components. Nevertheless testing the system in general becomes more challenging. Large number of integrational tests should be implemented to verify that the system is correctly working. Moreover more resources will be needed to test the system. Well designed continuous delivery pipeline is a must in this case.

Code. Developers are allowed to choose different technologies, frameworks , libraries for the modules that they are working on. The size of the team working on particular module might be relatively small, therefore if one member leaves the company a developer from another team will have to learn new tool, before he/she can start to contribute.

Security. Modules can be reused by other services. Modularity gives hacker more entry points to penetrate in the system. Security is a must when handling user's data. Thus a lot of effort should be put to handle security risks in microservice architecture. [5]

Design. Large number of available technologies provides different ways of achieving the same result. Hence well thought choice should be done when designing the architecture of the application.

Nowadays microservices is a trend in software development. There are crucial benefits that make usage of microservice architecture advantageous. On the other hand it has

some disadvantages that were covered in this section. Main challenge is that this design brings complexity, although it could be overcomed with well thought decisions and the amount of effort that company put in the process of development of a product.

2.2 Cloud computing

Term cloud computing generally means the delivery of access to the compute resources via the internet. The main advantage it provides is that user does not need to build and maintain own infrastructure and can focus more on developing an application. Cloud computing started to boom a decade ago. The first big player was Amazon, with its product called Amazon Web Services, which was released in 2002. In 2006 Amazon introduced *Elastic Compute Cloud* that gave users opportunity to rent computing resources on which they could run their applications. In 2009 another two big players entered the market: Google with *Google App Engine* and Microsoft with *Windows Azure*. There are three types of service models traditionally distinguished in cloud computing.

Software-as-a-Service – a model in which companies provide access to their software that is working in a cloud infrastructure [6]. Typical example is Microsoft Office 365. One can utilize applications such as Word, Excel, PowerPoint online and have access to them through different devices. Benefit of using this type of service is that user does not need to install any software . Moreover backup of data is responsibility of cloud provider.

Platform-as-a-Service - a model, in which user gets opportunity to utilize cloud infrastructure with chain of tools to manage applications. Management of physical or virtual compute instances, networking, storage is carried out by cloud provider. [6]

Infrastructure-as-a-Service - the main difference between PaaS model is that in IaaS user gets access to cloud infrastructure. It provides more possibilities to manage resources, control networking, install various software. At the same time physical layer is still responsibility of cloud provider. [7]

Nowadays there are more models apart from those that were explained earlier. For instance Desktop as a Service DaaS which gives ready to use workspace for the client. Nevertheless they are not part of this survey. [8]

Another topic that should be covered in this section is the types of clouds. First one is Public Cloud. This type of cloud is available for general public. Main drawback of it is lack of security. Second one to be mentioned is Private Cloud. In general this cloud is used only within one organization. The last but not least is Hybrid Cloud. This is combination of the public and private ones.

2.3 Kubernetes

Understanding of containers is needed in order to have a better grasp of kubernetes. Previously applications were deployed on host systems, which leads to entanglement of application configurations, lifecycles, and with host operating system. Container image is a stand-alone software that has everything needed to execute. They are not coupled together and have their own file system. Moreover containers do not rely on host operating system, thus they could be easily deployed on various servers. Common practice to pack one application in one container image. This gives a great advantage in deployment speed. In continuous delivery containers help to make quick builds, releases, rollbacks. [9]

Container is the way to package application and to deploy on different environments. There is still a question of how to manage, scale ,recover them. Kubernetes tries to address these issues. Kubernetes is an open-source system for management of containerized applications powered by Google.

Kubernetes cluster is a cluster of compute nodes that act as one unit. It consists of two types of nodes: masters and nodes. Master is a management unit, that is responsible for scheduling, scaling, terminating, updating applications. Nodes are responsible for running applications. Figure 1 illustrates kubernetes cluster.

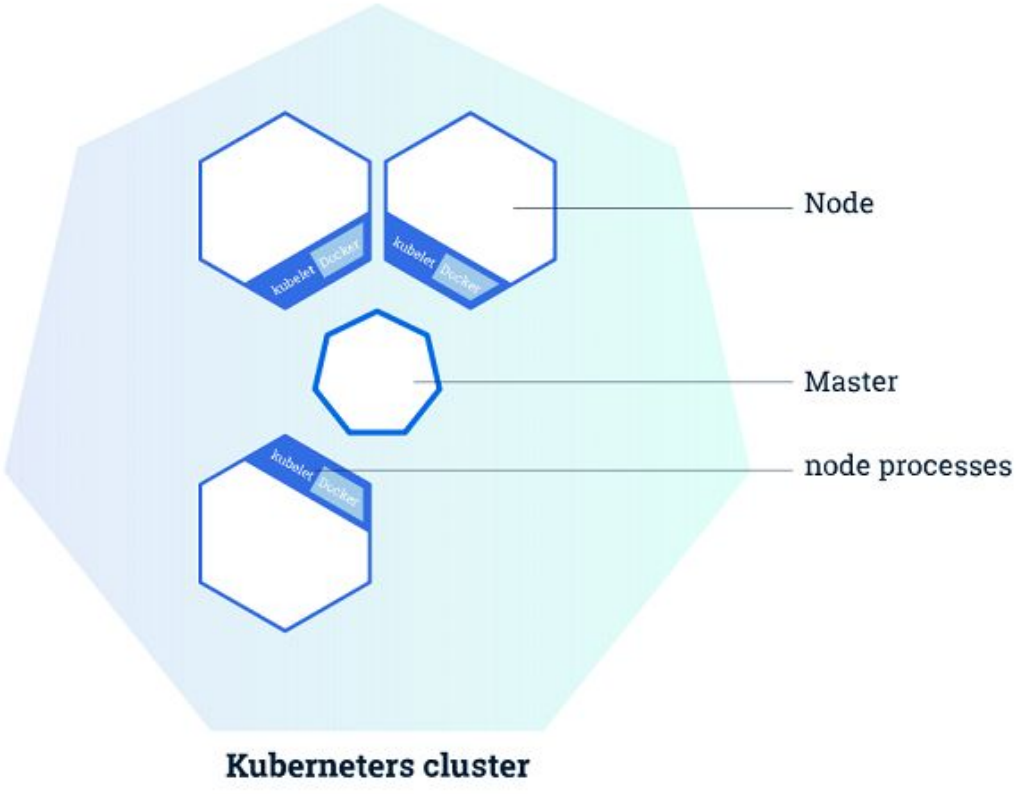

Figure 1. Kubernetes cluster. Reprinted from kubernetes documentation [9]

Each worker node has kubelet. It is responsible for communication with master. Moreover nodes need tool to create, run, delete container applications. It could be docker, rkt, LXD. Master exposes APIs to communicate with workers.

One can deploy containerised application in kubernetes cluster using *Deployment*. In kubernetes *Deployment* configuration describes how to update and create application. When one creates *Deployment* kubernetes generates *Pod*. It is an abstraction mechanism to represent one instance of application and some resources that coupled with it. When *Pod* is created it resides on some node till it is terminated. If node where the *Pod* was running goes down a new one will be automatically scheduled in another node. There might be multiple Pods running on the same node depending on node's resource limitations. Master is responsible for scheduling them across all available nodes. A worker could be a physical server or a virtual machine.

Another abstraction that is needed to understand kubernetes is a *Service*. It is an abstraction that defines relation of logically connected *Pods* and the ways to access them. *Services* allows access to applications in different ways, which is defined by the type of the *Service*.[9] The following types are available:

- ClusterIP (default) - this type makes application available only within cluster
- NodePort - makes a *Service* reachable outside the cluster
- LoadBalancer - create load balancer and assigns fixed IP to *Service* .

Figure 2 illustrates an example of kubernetes cluster. One can see three nodes and two services. Service A has only one pod, whereas service B has three which are coupled together. In the middle of the picture master is illustrated that holds information about the Services.

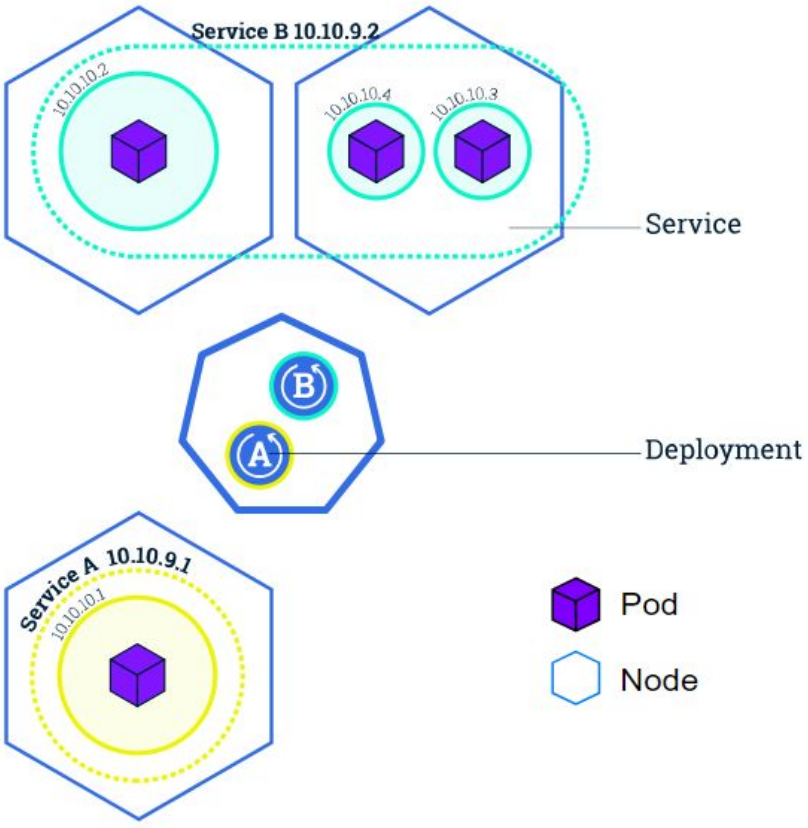

Figure 2. Example of kubernetes cluster

Main advantage of kubernetes cluster is an easy way to scale in and scale out *Pods*. It could be easily achieved by changing the number of replicas of a *Pod*. When master scale up application it creates a copy of *Pod* in one of the nodes with enough resources. Running multiple replicas of one *Pod* requires mechanism to distribute traffic. This is handled by the Service associated with that applications.

2.4 Continuous integration and continuous delivery

The high competition in the market compel companies to allocate numerous human and computing resources to implement continuous practices. These practices provide several benefits for software development. To begin with frequent releases. This improves customer experience because new functionality is delivered faster [10]. On the other hand companies and developers receive quick feedback about their product . Moreover connection between operational team and development team is enhanced and the number of manual tasks is decreasing.

Continuous Integration is a software development practice in which developers commit frequently, usually on daily basis. Each integration is checked by the automated test system to catch failures in early stages. There are several common practices that needs to be taken into account when discussing CI. They were introduced by Martin Fowler and they will be described below. [11]

Maintain a Single Source Repository. The practice implies that a team uses version control system to maintain code. The main idea is to have one place with all needed components to build the project. Therefore a project should be build without any additional steps or dependencies when new developer downloads a code from repository.

Automate the Build. There are variety of build tools available. *Make* is a commonly used tool for these purposes. Nowadays *bazel* also gains popularity, because of its rich functionalities. The key concept of this practice is to have one command that could build the system. It could consists of different stages like: compiling code, unit and

integration testing ect, but build should happen without any additional manual steps. [12]

Make the Build Self-Testing. Testing should be well thought and designed and if any of the tests fails the build should fail as well. Enough information should be provided by build tools to the developer to explain the reason for failure.

Everyone Commits to the Baseline Every Day. Developer should be encouraged to make smaller commit more oftenly. Therefore more rounds of CI will be executed and more errors could be detected [11].

Keep the Build Fast. The faster a developer gets the feedback from the CI system the earlier he would fix it. Main point is to keep build pipeline updated.

Test in a Clone of the Production Environment. “Staging” environment is beneficial, because one keeps main line with less errors. Although implementation of test environment requires more resources and time.

Make It Easy to Get the Latest Deliverables. One should consider a way of getting product packages easily for all developers, for instance by maintaining artifact repository.

Everyone can see what's happening . All developers should see the status of builds, history of CI runs, thus there will be a clear picture of current issues. Jenkins web interface is an example of how to achieve that.

Automate Deployment. There might be multiple testing,building environments in one project. Some failures could occur in these environments or developer would need to create additional one. The good practice is to have scripts that helps to automate this process.

Continuous delivery is the extension of the process of continuous integration, which provides a sustainable way of releasing product. This implies that not only process of testing, building should be automated, but also the process of release. In order to start

developing this process one need to have stable and reliable continuous integration. This process has a manual step of deploying product to production, nevertheless deployment should be automated. The benefit of continuous delivery is that you have frequent small releases. Thus one can deliver small functionalities or fix bugs faster.

Continuous deployment is an enhancement of continuous delivery. In this process there is no human interaction in a production pipeline. This requires to have extremely reliable testing, constantly updated documentation. As a result company gets constant small releases, that provide fast feedback and increasing quality of a product. Figure 2 illustrates relations of all the continuous practices described in this chapter. [13]

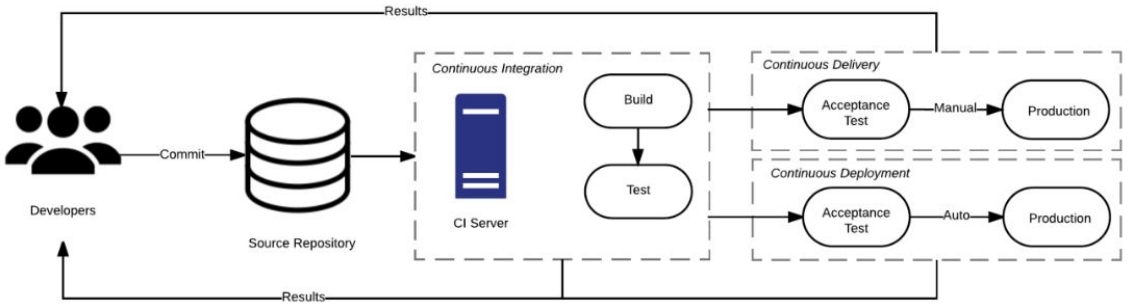

Figure 2. The relationship between continuous integration, delivery and deployment. Reprinted from [10]

3 Project methods and materials

3.1 Case summary

The company is developing a solution that provides end user with kubernetes cluster platform. One can deploy kubernetes on various platforms: bare metal servers, virtual machines or event on laptop. The solution is used to install kubernetes cluster with all the needed components: monitoring, logging, networking, authentication ect, on a set of pre-provisioned servers. Hitherto most of the effort was put into developing new features for the product. Due to the fact that after trials product gained interest a decision to increase the number of developers was made. In order to satisfy all the needs of a new team an advanced testing pipeline should be implemented.

Current implementation consists of the following steps:

- Picking up new commit
- Building images
- Deploying kubernetes on servers
- Testing
- Publishing results

There is a limited number of environments where the stage of kubernetes deployment could be held. Each kubernetes cluster requires at least 6 nodes. One node is needed for load balancer, three more nodes for masters to have high-availability solution and at least two more nodes for workers. This could become a potential bottleneck in a large team. Due to the fact that most of the developers work at the same time the utilization of test resources could be the problem that a developer would have to wait for finishing of another's commit test execution. In order to overcome this issue it was decided to take Openstack into use. The description of this service will be introduced in the next section. The main advantage of Openstack is that it provides an opportunity to spawn virtual machines from the pool of available resources.

As was mentioned earlier the main challenge is to provision enough clean instances for kubernetes cluster. Figure 3 describes the core components that are in the product.

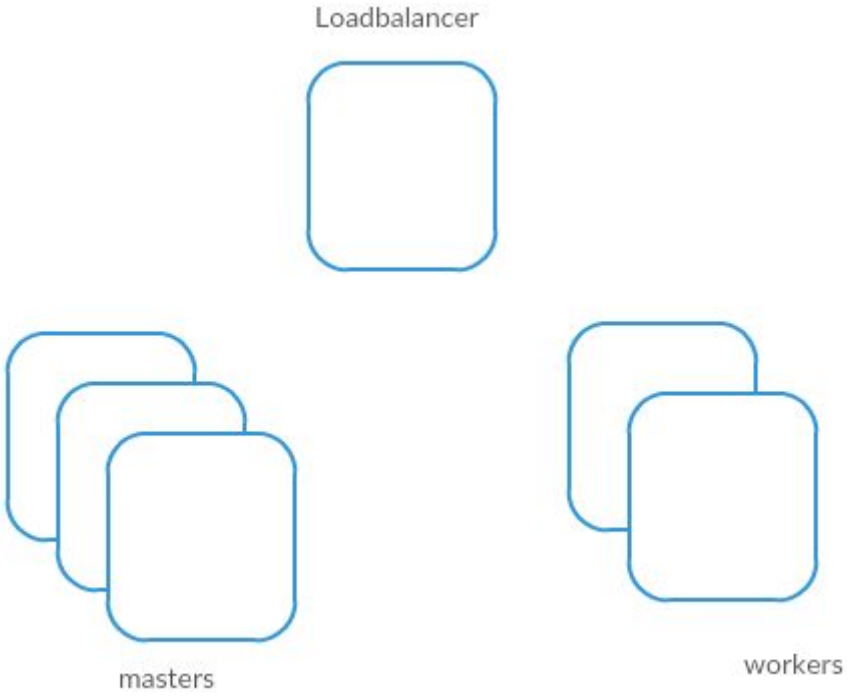

Figure 3. Product structure

3.2 Technologies

3.2.1 Openstack

Openstack is a set of tools for manipulating pools of resources. It enables possibility for managing public or private clouds. Openstack is one of the largest open source projects that is driven by Openstack Foundation. It is free and released under Apache license 2.0. One can easily take it and deploy it. Nevertheless deployment of Openstack requires high competence in this field. Moreover a couple of engineers should be allocated for managing the system, because it consists of different components which could produce errors from time to time.On top of that Openstack project is being constantly developed and some updates might be needed. If the company is not willing to manage Openstack itself then there are multiple possibilities

to rent resources from various providers: MIRANTIS, PLATFORM9, ZEROSTACK to name a few. [14]

Main use case for the openstack is the creation of virtual machines. This possibility enables user to deploy new services faster. Moreover one can easily scale up or down instances. For example if the number of users of an application is growing a new back-end server could be brought up to divide traffic.

Developers can access Openstack through diverse application programming interfaces. There are two ways of accessing them. The first one is through the web interface. It provides coherent layout to manipulate Openstack. One can see list of created virtual machines, networks, volumes. Moreover a developer can delete, create all of the available resources in a particular project. Admin in openstack has more privileges. Resource limits , project creation, adding new users is done by the admin. Nevertheless web interface has a lot of limitation. Second option to access Openstack is through client command line interface. It is a python library that has common structure to call all the components of the Openstack. [15]

Openstack consists of multiple components. Each of them is an independent and open source project. Relations between components could be observed from the Figure 4. There are core components that are needed for deployment and they are maintained by Openstack community. [16]

- Nova is in charge of managing virtual machines and other instances.
- Cinder is a block storage. It provides persistent storage for virtual machines.
- Neutron enables networking. It is responsible for communication between instances.
- Horizon is a graphical interface to access to Openstack
- Keystone is in charge of authentication and authorization. A list of users with access rights and ways to manipulate it.
- Glance is a storage for images. Image is virtual copy of hard disk that is used to spawn instances.
- Swift is an object storage in Openstack. Commonly used to store media files and backups.
- Ceilometer provides information of how much user consume resources.

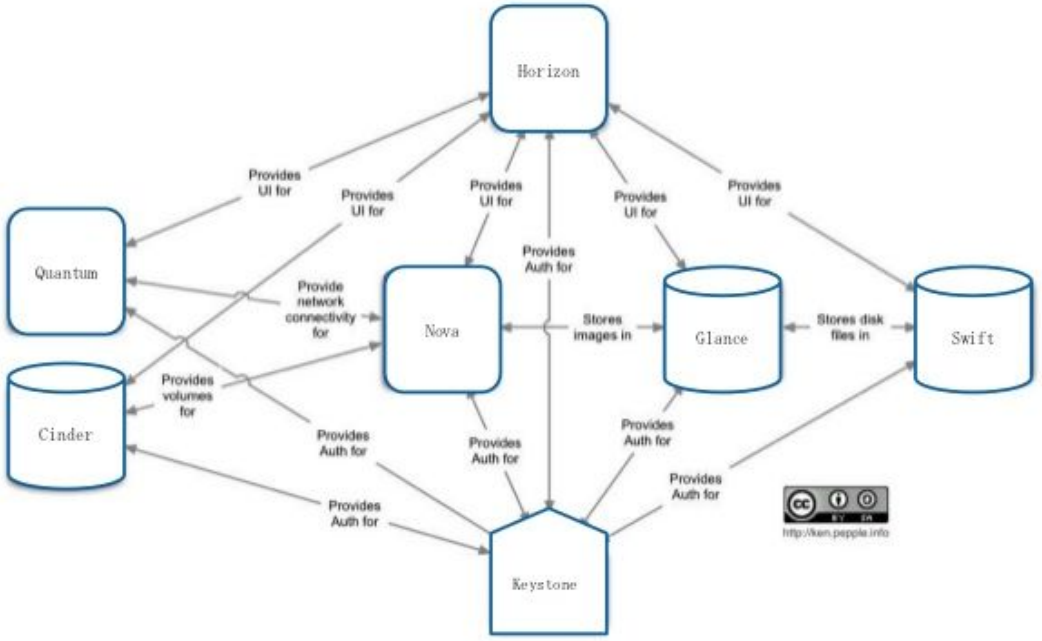

Figure 4. Openstack component relations. Reprinted from [16]

3.2.2 Comparison of heat and ansible

Two ways of creating infrastructure for the Openstack were considered for this project : ansible and heat.

Heat

Heat is a service to create infrastructure in the Openstack using templates. Template is a human readable code that describes cloud application. All the relations between resources are in the templates and heat creates them in order by calling different Openstack application programming interfaces. Moreover there are various plugins that can customise heat possibilities. [17]

Listing 1 illustrates heat template example that creates specified number of servers in the Openstack. Typical structure of heat template is that in the beginning one gives parameters that will be used for resources. They might be defined or come from other resources. Section resources describes what will be created by Openstack. In this particular case six servers will be created in openstack with a *CentOS-image* that is

stored in glance, with a *m1.small* flavor, and *test-network* that was pre-created in this project.

```
heat_template_version: 2014-10-16
description: Template create specific number of vms in Openstack

parameters:

   instance_count:
       type: number
       description: Number of instances
       default: 6
   image_id:
       type: string
       description: Image to be used (RHEL/Centos 7 compat) for base OS
       default: CentOS-image
   instance_flavor:
       type: string
       description: Type of instance (flavor) to deploy for the instance node
       default: m1.small
   network_id:
       type: string
       description: ID of the public net
       default: test-network
resources:

   instances:
       type: OS::Heat::ResourceGroup
       properties:
           count: { get_param: instance_count }
           resource_def:
               type: OS::Nova::Server
               properties:
                   image: { get_param: image_id }
                   flavor: { get_param: instance_flavor }
                   networks: [{network: { get_param: network_id}}]
```

Listing 1. Heat template to create servers *heat.yaml*

In order to execute the template from listing 1 one needs server with access to Openstack and to run command from listing 2.

```
openstack stack create --wait -t heat.yaml alex-testing
```

Listing 2. Heat command to create infrastructure

Ansible

Ansible is a software used to configure systems, deploy various libraries. It is commonly used in continuous configuration automation [18]. Even though ansible is primarily used for configuration purposes it has a cloud module for Openstack. This module provides opportunity to call Openstack APIs. One can achieve the same goal using either ansible or heat [19]. Ansible was taken into consideration, because most of the developers in the team obtain experience with this software. Listing 3 illustrates example of deploying virtual machines in Openstack.

```
---

- name: Deploy vms on OpenStack
  hosts: localhost
  gather_facts: false
  vars:
      servers:
        - name: server1
          image: CentOS-image
          flavor: m1.small
          network: test-network
        - name: server2
          image: CentOS-image
          flavor: m1.small
          network: test-network
        - name: server3
          image: CentOS-image
          flavor: m1.small
          network: test-network
        - name: server4
          image: CentOS-image
          flavor: m1.small
          network: test-network
  tasks:
    - name: launch instances
      os_server:
        name: "{{ item.name }}"
        state: present
        image: "{{ item.image }}"
        flavor: "{{ item.flavor }}"
        network: "{{ item.network }}"
      with_items: "{{ servers }}"
```

Listing 3. Ansible playbook to create servers *ansible.yaml*

One can deploy infrastructure using ansible with the command from the listing 4.

```
ansible-playbook ansible.yaml
```

Listing 4. Ansible command to create infrastructure

Conclusion

Two ways of orchestrating infrastructure in Openstack were considered in this section heat and ansible. Heat is an official approach to work with Openstack. On the other hand ansible is commonly used in automation tasks and could easily implemented to manipulate Openstack. [19]

The main advantage of using heat is that when the stack is created using command from listing 2, all the information about allocated resources is storedt in Openstack. In order to delete stack one can execute one command that is mentioned in listing 5 and Openstack will free all the allocated resources.

```
openstack stack delete alex-testing
```

Listing 5. Heat command to delete infrastructure

In order to free resources that were created using ansible some extra work must be carried out. To begin with a developer will need to keep track of allocated resources . Usually it is placed in the server from which ansible is executed. An issue might occur if the server with the list of resources is used for other purposes, data corruption could happen, because of other processes. Moreover a separate playbook is needed to free all the resources retrieved from Openstack. Listing 5 illustrates simple example with only servers created by the Openstack. In the real scenario there is a need to create networks, key paris, floating ips , servers to name a few. Therefore deletion of all resources should be handled in a separate ansible playbook in a proper order.

Heat does not have ansible disadvantages that were mentioned in previous paragraph. Consequently is chosen for the task of creating infrastructure in Openstack. The time that will be spend to study this technology will be compensated by its advantages.

3.2.2 Continuous Integration Tools

In general a developer makes at least one commit per day. Even one line change could lead to failures. In order to reduce integrational risks each team adopts various testing approaches. Continuous Integration is a practice in software development in which changes made to the project are automatically build,deployed,tested. Main pain point in CI is the time spend between the moment developer pushes the code and till the CI finishes its full circle. There are different continuous integration tools that are available. In the following section some of them will be described and decision on which one to use will be made.

Jenkins

Jenkins is a mature project that was started in 2004 by Kohsuke Kawaguchi initially called Hudson . It is an open source automation server. It is written in Java and supports commonly used version control systems CVS, Subversion, Git, Mercurial to name a few. [20]

There are numerous plugins that enhance functionality of Jenkins. For instance Jenkins Pipeline. It was release with 2.0 version. It allows a developer to put multiple builds in a single script. With that functionality in place it is easier to control the flow of the script and make modification to it. Moreover visualisation of the CI with this plugin enriches developer experience. Another plugin that worth to be mentioned is Nested View. It helps to organize jobs, which is especially useful in large project. [21]

TeamCity

TeamCity is a well known CI server that is developed by JetBrains. Developer can utilize all features of this product for free to some extend. One can have up to three build agents and define 100 jobs [22]. Moreover it has integration with modern development platforms. There are multiple publicly available plugins for TeamCIty. It support runners for Java, .NET by default and there are much more available from the plugin list. [23]

Travis CI

Travis CI is one of the oldest solutions available. It is free for open source projects. There are four pricing plans available for private projects.

Processes are described in .travis.yml file. There are different practices to speed up the build that are described in the official documentation. The easiest one is to use build matrix and to parallelize test cases. This could be achieved if the project is well structured and it does not have a lot of cross-dependencies. Moreover there are cache files available that help to decrease the configuration time before running tests. [24]

Conclusion

There were three technologies described in this section. They were chosen from the most popular solutions. Figure 5 illustrates results of the survey that were held among 1000 respondents in this field. Moreover experience and preferences of the current team were also taken into account. Jenkins was chosen as a solution because it is free self-hosted server that has all the required functionalities. Moreover all most developers from the team had previous experience with this tool

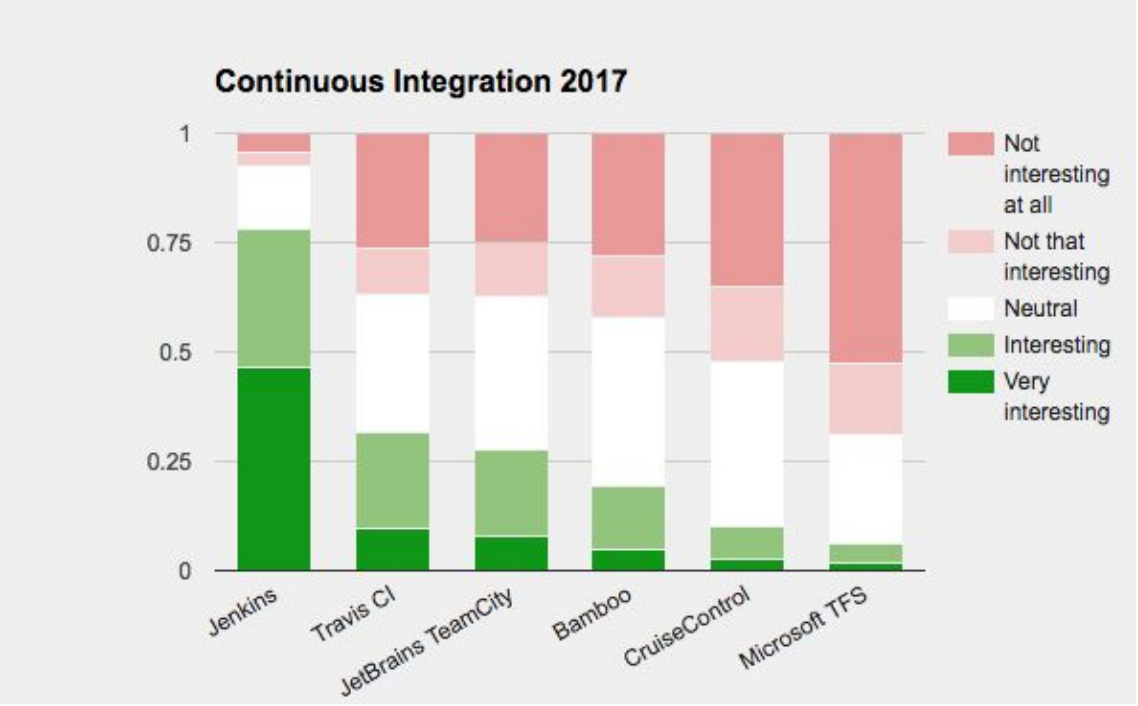

Figure 5. Continuous integration tools survey. Reprinted from [25]

4 Implementation

4.1 Solution design

Implementation pipeline consists of several stages. This section will discuss all the steps and new suggestions will be outlined. Figure 6. illustrates new implementation architecture.

Initially a developer pushes new commit to version control system. In order to run changes through CI one needs to receive review from another team member. This step could be also done after running through CI, but if a reviewer requests some changes then another round of tests will be needed. After the code was reviewed a developer could start testing his changes.

Automated process picking up new commit. It is done by *Jenkins master* server which is illustrated in Figure 6. During this stage one of the available environments will be picked by the master for instance *CI-env-01* and code will be uploaded to the *minion*-01 which is situated in that environment.

Next is the stage in which images are being build. There are many components in the kubernetes cluster and one of the requirement to build them before releasing the product. There is cache therefore if there are no changes in the code for the images then this step will be skipped. There are separate jenkins servers dedicated for that purposes.

After that new servers will be spawned in Openstack. That is one of the main changes in new implementation. In the Figure 6 all the new servers are illustrated with dashed green lines. It means that changes will be tested on fresh instances. It is an important change, because previously changes were tested against machines that were cleaned up by the script.This is an error prone way of testing. At this stage errors might occur with Openstack. Heat templates that will create all the infrastructure for the cluster produce all of API requests. Therefore there might be connection errors, timeouts , internal Openstack failures.

Afterwards a kubernetes will be installed on the freshly created virtual machines. During this stage the product is installed. Ansible is used as the main tool to deploy kubernetes cluster. This process could reveal different errors, because various failures could occur during ansible configuration.

Whenever a new feature is added to the product a developer needs to write test to verify it does not break other functionality. They are checked during the next testing stage. Robot framework is used as a main testing tool.

Publishing results is the last stage. In this stage results of the tests will be available for the developer in the jenkins view.

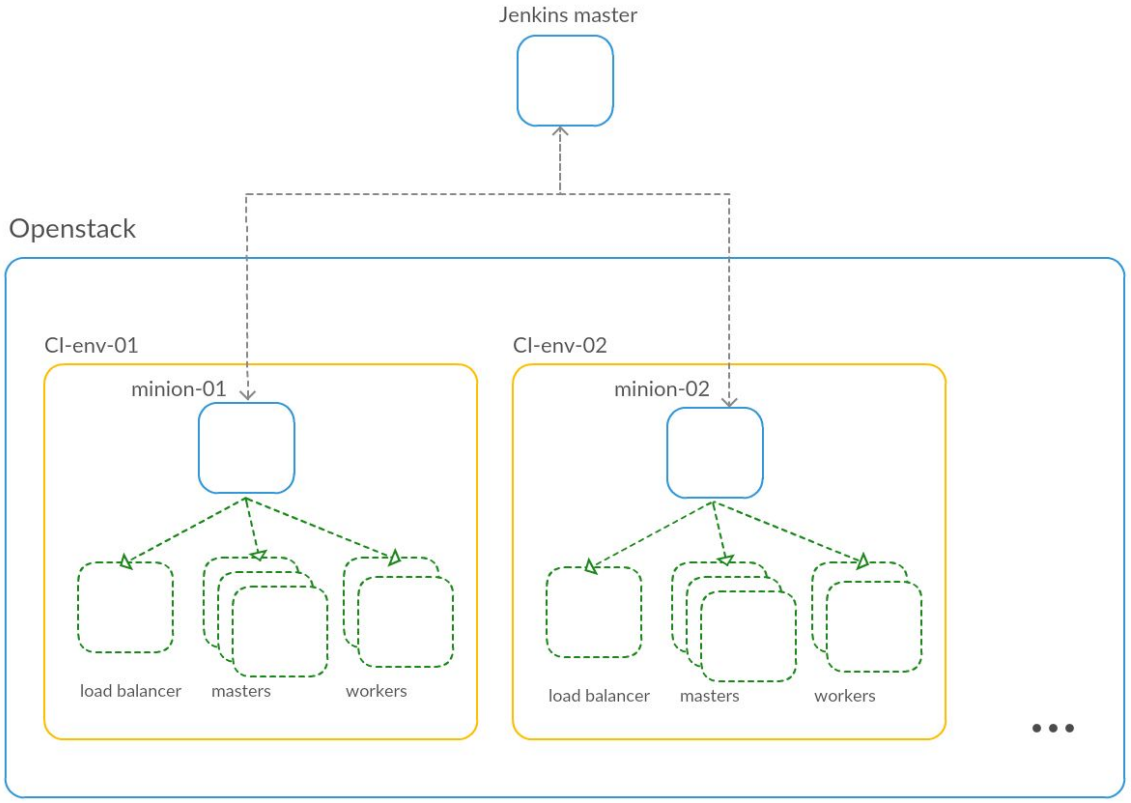

Figure 6. Implementation design

4.2 Openstack minion setup

In web interface for the Openstack one can easily create virtual machine that will be used as minion. During the process of host creation a developer provides the public key of his working station. Subsequently it will give possibility to ssh to that machine. Moreover floating ip needs to be attached to newly created host in order to have access to it. Floating IP is a publicly routable IP. Users of Openstack have to attach this IP to their instance in order to have external access [26]. Pool of floating IPs is managed by Openstack manager [26] . If there were no internal failures in Openstack then the instance will be created and its state will be *Running* as could be seen from the Figure 6.

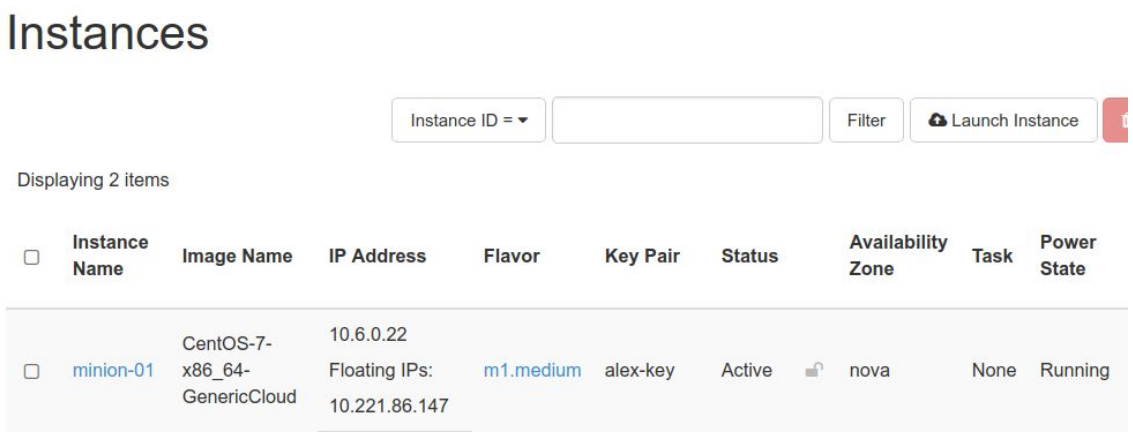

Figure 6. Running instance in Openstack

After the virtual machine is created a developer has a host with a clean operating system. In this case it is CentOS. The CentOS is a free operating system driven by the open source community, which equipped with wide range of security features [27]. In order to make this machine a minion that would be working in automation pipeline additional configuration is required. In the beginning there are only two projects created for the CI purposes, therefore configuration of two minions is demanded. Moreover in future number of projects will grow, thus configuration of minions should be automated.

Bash script could be a suitable solution for automating node configuration. Listing 6 illustrates the implementation of the script. Main function of the scripts describes the flow of the script. Function implementation is hidden not to clutter up here an example and they will be explained briefly later.

_install_libraries function installs missing libraries for the operating systems. Examples of libraries needed to run openstack client:

- python-openstackclient
- python-heatclient

_create_stackrc_file creates the file that holds variables needed to execute openstack client requests. The following variables must be included and defined in that file:

- OS_PROJECT_ID=
- OS_REGION_NAME=
- OS_USER_DOMAIN_NAME=
- OS_PROJECT_NAME=
- OS_IDENTITY_API_VERSION=
- OS_PASSWORD=
- OS_AUTH_URL=
- OS_USERNAME=
- OS_INTERFACE=

_create_custom_user . There is a need to create user in operating system with a predefined name.

_connect_to_master established connection to jenkins master. There are couple of ways to achieve this goal. First one is to launch it from master using ssh. To do that one would need to copy public key of master to *~/.ssh/authorized_keys.* Second option is to launch agent using Java Network Launch Protocol. It provides an opportunity to launch agent as UNIX daemon and it is considered to be more secure, because of these reasons it was chosen for the implementation. [28]

```
#!/usr/bin/env bash
set -uex
```

```
#Function implementation
# ...

_finish() {
   if [ $? -ne 0 ]; then
       echo "Failure occur during minion setup"

       _clean_up

   else
       echo "Minion setup finish with Success"
   fi
}

### main
main()
{
   trap _finish EXIT

   _install_libraries

   _create_stackrc_file

   _create_custom_user

   _connect_to_master

}

main "$*"
```

Listing 6. Script to configure minion *minion_setup.sh*

The last point to be discussed in listing 6 is the usage of *trap.* While running a script user may encounter various errors. The trap gives an opportunity to catch an interrupt signal and to perform clean up procedure.

4.3 Openstack infrastructure heat template

Typical structure of heat templates was introduced in listing 1. In the Figure 3 illustrates infrastructure that needs to be created by the Openstack. Therefore 3 masters, loadbalance, and two workers are required.

Workers

In the current implementation workers are located in the same network as all other kubernetes nodes and CI minion. All the parameters are precreated and allocated using *get_param* which could be seen from the listing 7. Resource Group types in Openstack enables creation of several identical instances [29]. *Key_name* in listing 7 is a public key of CI minion, which provides ssh access to this host.

```
workers:
 type: OS::Heat::ResourceGroup
 properties:
   count: {get_param: number_of_workers}
   resource_def:
     type: K8S::Worker
     properties:
       name:
         str_replace:
           template: worker-%index%
       flavor: {get_param: master_flavor}
       image: {get_param: image}
       key_name: {get_param: key_name}
       username: {get_param: username}
       network: {get_param : cluster_network}
       subnet: {get_param: cluster_subnet}
       security_group: {get_param: security_group}
```

Listing 7 Worker setup in heat template

Masters

The difference between master and worker definition from the heat point of view is that they have virtual ip port see listing 8. Nevertheless it is highly important, because it is

needed for high-availability kubernetes masters. Virtual IP is an entry point for a set of master servers. Using third party library called *keepalive* one cat setup high-availability implementation, where this software will choose one of masters to act as main and all others to become backup servers.

```
masters:
 type: OS::Heat::ResourceGroup
 properties:
   count: {get_param: number_of_masters}
   resource_def:
     type: K8S::Master
     properties:
       name:
         str_replace:
           template: master-%index%
       flavor: {get_param: master_flavor}
       image: {get_param: image}
       key_name: {get_param: key_name}
       username: {get_param: username}
       network: {get_param : cluster_network}
       subnet: {get_param: cluster_subnet}
       security_group: {get_param: security_group}
       vip_port: { get_attr: [vip_port, fixed_ips, 0, ip_address]}
```

Listing 8. Master setup in heat template

Load Balancer

Currently in design implementation only one Load Balancer is needed, nevertheless in future it might be required to have high-availability for this service. The difference between worker implementation is that Load Balancer has floating IP attached to it. It is needed, because all the incoming traffic could reach nodes only through floating IP.

The result of running heat templates from minion-01 with the command from listing 2 could be observed in Figure 7. It illustrates the view of web interface of Openstack. One can see a list of running servers that were created using heat and minion from which it was executed. The same information could be retrieved from minion using openstack command line interface for instance using command from listing 9. Moreover the command will provide more data, which is mostly needed by Openstack admins.

```
openstack server list
```

Listing 9. Command to retrieve server list information from Openstack

Instance Name	Image Name	IP Address	Flavor	Key Pair	Status	Availability Zone	Task	Power State
load-balancer	CentOS-7-x86_64-GenericCloud	10.6.0.25 Floating IPs: 10.221.86.129	m1.medium	minion-01-key	Active	nova	None	Running
master-3	CentOS-7-x86_64-GenericCloud	10.6.0.13	m1.medium	minion-01-key	Active	nova	None	Running
master-2	CentOS-7-x86_64-GenericCloud	10.6.0.23	m1.medium	minion-01-key	Active	nova	None	Running
master-1	CentOS-7-x86_64-GenericCloud	10.6.0.10	m1.medium	minion-01-key	Active	nova	None	Running
worker-2	CentOS-7-x86_64-GenericCloud	10.6.0.12	m1.medium	minion-01-key	Active	nova	None	Running
worker-1	CentOS-7-x86_64-GenericCloud	10.6.0.19	m1.medium	minion-01-key	Active	nova	None	Running
minion-01	CentOS-7-x86_64-GenericCloud	10.6.0.22 Floating IPs: 10.221.86.147	m1.medium	alex-key	Active	nova	None	Running

Figure 7. Instances list in Openstack web interface

4.4 Jenkins pipeline

There are several plugins that have to be in installed in order to have full functionality. Jenkins Pipeline is a set of plugins that enables the implementation of *continuous delivery pipeline* into Jenkins. *Continuous delivery pipeline* is the definition of your processes that a commit will go through before it gets merged. It is written in a separate Jenkinsfile, which could be a part of a source control repository. Therefore it could be reviewed and edited by other team members. Pipeline domain-specific language syntax is used to write Jenkinsfiles. There are two types of syntax declarative and scripted. Declarative pipeline syntax is the latest and has more functionalities that scripted one. Thus it is used in this project. Example of pipeline with declarative syntax could be seen in listing 9. [30]

```
pipeline {
    agent {
        label 'kubernetes-platform-check-verification'
    }
    options {
        timestamps()
    }
    environment {
         ..
    }
    stages {
        stage('Prepare Code') {
            steps {
              ..
             }
        }
        stage('Build images') {
            steps {
                ..
            }
        }
        stage('Create VMs') {
            steps {
                ..
            }
        }
        stage('Deploy kubernetes') {
            steps {
                ..
            }
        }
        stage('Testing') {
            steps {
                  ..
            }
        }
    }
    post {
        always {
           ..
        }
    }
}
```

Listing 9. Jenkins pipeline example

Key concepts of jenkins pipeline that are introduced in listing 9:

- Agent - server that is able to execute pipeline
- Stage - defines sets of task that are coupled together
- Steps - tasks that should be accomplished by jenkins

Jenkins can execute various jobs, which could be configured using web interface. In the beginning one clicks *New item* , then enters name and chooses one of the items from the list of jobs. For this project *Pipeline* is used. After clicking OK user is moved to the configuration of the job. During this stage trigger events are configured. Moreover Jenkinsfile is included at this stage. Two ways to add Jenkinsfile to the jenkins job: reference to the file in the source code of the project or hardcode it in web interface. Another noticeable advantage of Jenkins pipelines is that a developer can run it manually , pause , edit. Moreover additional plugins allows to analyze results of execution. Figure 8. illustrates one round of CI to test kubernetes platform.

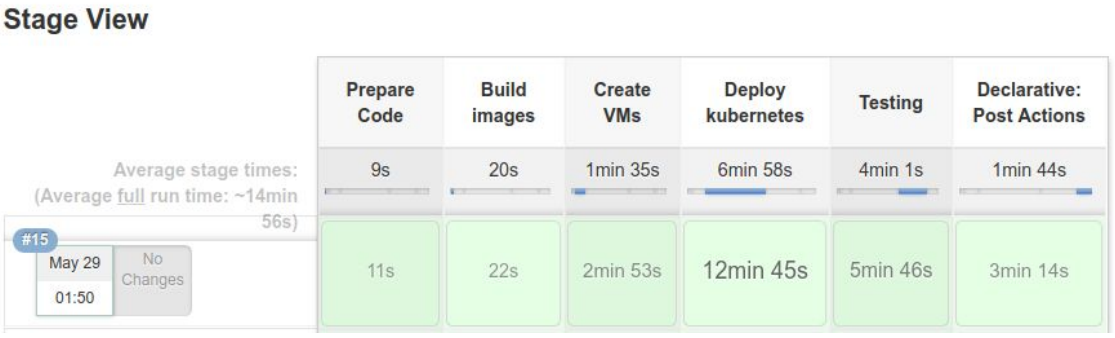

Figure 8. View of pipeline execution result in jenkins web interface

5 Results

5.1 Summary

To sum up the results,an enhanced implementation of CI pipeline was integrated for the cloud platform continuous integration. Extensive resources are required to test the kubernetes-based cloud platform. In current implementation at least 6 virtual machines are spawned to form kubernetes cluster. The infrastructure for kubernetes is created using heat templates every round of CI with an average time of 3 minutes. Thus the product is always being tested on clean servers.

The number of developers is growing in the team. Therefore in order to achieve the delivery without delays more CI environments needed. The new implementation satisfies this requirement fully, because Openstack allows dynamic creation of new servers. Moreover the projects in Openstack could be as well used for development purposes.

Script to set up minions in the project supports future deployments of new environments. Moreover there could Openstack failures occur, therefore scripts helps to automate the process of creation.

Jenkinsfile that describes the whole workflow of pipeline is now located in source code folder of the product. This makes automation more reusable and editable, because all team members could contribute. Moreover Jenkinsfile can be used as a basis for other jenkins jobs, which makes this code highly reusable.

Figure 8 illustrates the view from jenkins web interface. As one notices all the stages are separated and it is easier for the developer to see where the failure occurs. Moreover jenkins provides the possibility to run build again or to pause it.

5.2 Future improvements

There are several improvements that could be added to the proposed implementation. To begin with an automation of Openstack project creation could be added. Currently only scripts that supports minion automation is in place. There is possibility add automate the creation of the whole environment in Openstack. Another improvement that worth mentioning is staging environment. Every new feature that is added will be tested at first in a staging environment before merging to master branch.

6 Conclusion

Application development consists of various stages. In the begging developers gather information and make prototypes. After all architectural decisions are made the development phase starts. Each day developers make commits to code repository of the product. It is a common practice that in order to merge any code to the main branch it should go through circle of tests and to be review by other members of the team. This process should be automated, because it would help to have frequent releases. There are several continuous integration and continuous delivery practices that support automation of this process. They are described in the second chapter of this thesis. Moreover it contains theoretical studies that are needed for better understanding of the project that is being developed.

The main goal of this thesis was to develop an automated pipeline that would help to facilitate all the demands of project. The implementation should follow the guidelines of the continuous integration practises. The solution pipeline fulfils all the requirements of for this product. Different tools very used to achieve the goal of the project, such as Jenkins, heat, Openstack.

Nevertheless there are new CI technologies that are coming on the market, therefore this process should be always review and updated. Moreover there are still possibilities to automate some steps and to make the solution more generic. In conclusion even though there are several opportunities for improvement the objectives for this project were accomplished.

References

1. Top 50 Cloud Companies - Datamation [Internet]. Datamation.com. 2018 [cited 30 April 2018]. Available from: https://www.datamation.com/cloud-computing/cloud-computing-companies.htm
2. Bakshi K. Microservices-based software architecture and approaches [Internet]. IEEE Xplore Digital Library; 2018 [cited 5 May 2018]. Available from: https://ieeexplore.ieee.org
3. Newman, S. (February 2015). Building Microservices. USA: O'Reilly Media Inc
4. Shadija D, Rezai M, Hill R. Towards an understanding of microservices [Internet]. IEEE Xplore Digital Library; 2018 [cited 7 May 2018]. Available from: https://ieeexplore.ieee.org
5. Microservices: Benefits and Challenges - DZone Integration. [online] Available at: https://dzone.com/articles/microservices-benefits-and-challenges [Accessed 20 May 2018].
6. Bataev A, Rodionov D, Andreyeva D. Analysis of world trends in the field of cloud technology [Internet]. IEEE Xplore Digital Library; 2018 [cited 10 May 2018]. Available from: https://ieeexplore.ieee.org
7. What is cloud computing? | IBM Cloud. [online] Ibm.com. Available at: https://www.ibm.com/cloud/learn/what-is-cloud-computing [Accessed 19 May 2018].
8. Belbergui C, Elkamoun N, Hilal R. Cloud computing: Overview and risk identification based on classification by type [Internet]. IEEE Xplore Digital Library; 2018 [cited 10 May 2018]. Available from: https://ieeexplore.ieee.org
9. Kubernetes.io. (2018). What is Kubernetes? - Kubernetes. [online] Available at: https://kubernetes.io/docs/concepts/overview/what-is-kubernetes/ [Accessed 13 May 2018].
10. 9 Continuous Integration, Delivery and Deployment: A Systematic Review on Approaches, Tools, Challenges and Practices
11. Fowler, M. (2018). Continuous Integration. [online] martinfowler.com. Available at: https://martinfowler.com/articles/continuousIntegration.html [Accessed 16 May 2018].
12. dzone.com. (2018). Continuous Integration Part 3: Best Practices - DZone DevOps. [online] Available at: https://dzone.com/articles/continuous-integration-part-3-best-practices [Accessed 14 May 2018].
13. Atlassian. (2018). Continuous integration vs. continuous delivery vs. continuous deployment |. [online] Available at: https://www.atlassian.com/continuous-delivery/ ci-vs-ci-vs-cd [Accessed 15 May 2018].
14. OpenStack. (2018). Open source software for creating private and public clouds.. [online] Available at: https://www.openstack.org/ [Accessed 16 May 2018].

15. Opensource.com. (2018). What is OpenStack?. [online] Available at: https://opensource.com/resources/what-is-openstack [Accessed 16 May 2018].
16. Ismaeel S, Miri A, Chourishi D. Cloud Management Platforms: A Review [Internet]. IEEE Xplore Digital Library; 2018 [cited 12 May 2018]. Available from: https://ieeexplore.ieee.org
17. Docs.openstack.org. (2018). OpenStack Docs: Welcome to the Heat documentation!. [online] Available at: https://docs.openstack.org/heat/latest/ [Accessed 1 May. 2018].
18. Docs.ansible.com. (2018). Ansible Documentation — Ansible Documentation. [online] Available at: https://docs.ansible.com/ansible/2.3/index.html [Accessed 28 Apr. 2018].
19. Red Hat Stack. (2018). Full Stack Automation with Ansible and OpenStack. [online] Available at: https://redhatstackblog.redhat.com/2016/10/13/full-stack-automation-with-ansible-and-openstack/ [Accessed 7 May 2018].
20. CloudBees. (2018). About Jenkins. [online] Available at: https://www.cloudbees.com/jenkins/about [Accessed 9 May 2018].
21. Praqma.com. (2018). Top Jenkins plugins. [online] Available at: https://www.praqma.com/stories/top-jenkins-plugins/ [Accessed 25 May 2018].
22. JetBrains. (2018). TeamCity: Hassle-free CI and CD Server by JetBrains. [online] Available at: https://www.jetbrains.com/teamcity/ [Accessed 27 Apr. 2018].
23. Pecanac, V. (2018). Continuous Integration With TeamCity - Code Maze. [online] Code Maze. Available at: https://code-maze.com/continuous-integration-with-teamcity/ [Accessed 30 Apr. 2018].
24. Localytics. (2018). Best Practices and Common Mistakes with Travis CI. [online] Available at: https://eng.localytics.com/best-practices-and-common-mistakes-with-travis-ci/ [Accessed 18 May 2018].
25. JAXenter. (2018). Technology trends 2017: These are the most popular tools - JAXenter. [online] Available at: https://jaxenter.com/technology-trends-2017-these-are-the-most-popular-tools-132109.html [Accessed 1 Jun. 2018].
26. Siwczak, P. (2018). Floating IP for Networking in OpenStack Public and Private clouds. [online] Mirantis | Pure Play Open Cloud. Available at: https://www.mirantis.com/blog/configuring-floating-ip-addresses-networking-openstack-public-private-clouds/https://www.mirantis.com/blog/configuring-floating-ip-addresses-networking-openstack-public-private-clouds/ [Accessed 30 May 2018].
27. Davis, M. (2018). The Advantages And Disadvantages Of CentOS. [online] Future Hosting. Available at: https://www.futurehosting.com/blog/the-advantages-and-disadvantages-of-centos/ [Accessed 25 May 2018].

28. Wiki.jenkins.io. (2018). Distributed builds - Jenkins - Jenkins Wiki. [online] Available at: https://wiki.jenkins.io/display/JENKINS/Distributed+builds [Accessed 29 May 2018].
29. Wiki.openstack.org. (2018). Heat/Blueprints/scalable-resource-group - OpenStack. [online] Available at: https://wiki.openstack.org/wiki/Heat/Blueprints/scalable-resource-group [Accessed 30 May 2018].
30. Pipeline. (2018). Pipeline. [online] Available at: https://jenkins.io/doc/book/pipeline/ [Accessed 31 May 2018].