УДК 004.3

Г.Г. Киричек, канд. техн. наук, доц.,
Р.О. Котов, студент
Запорізький національний технічний університет
kirichek@zntu.edu.ua

Система автоматичного тестування та розгортання мережних сервісів

В роботі проводиться аналіз методів розгортання мережних додатків і пропонуються рішення, які націлені на скорочення часу та підвищення якості розгортання мережних сервісів в середовищі віртуальної приватної хмари, з використанням автоматичного тестування вихідного коду.

Ключові слова: хмарні технології, безперервна інтеграція, кластер, балансування навантаження.

Вступ

Користувачі мереж масово використовують програмне забезпечення як джерело контактів, розваг та додатків мережевих технологій для поширення необхідних послуг. Для підвищення економічної ефективності розробок сучасних сервісів потрібно проаналізувати етапи, які займають найбільше часу та скоротити витрати на них. Опираючись на відомості про життєвий цикл програмних продуктів, зроблено висновок про можливість автоматизувати такі процеси, як тестування та розгортання мережевих сервісів. Хоча існує певна кількість сервісів, які вирішують задачі автоматизації, вони в основному пропонують вирішення лише одного з цих завдань [1]. Тому прийнято рішення поєднати усі підходи в єдину систему, яка призначена для автоматичного тестування та розгортання мережевих сервісів.

Постановка задачі

Мета роботи – розробка системи для автоматичного розгортання мережних додатків після підтвердження успішного виконання тестових операцій, в результаті застосування якої зменшується час на отримання працездатної версії в середовищі віртуальної приватної хмари.

Організація процесу розробки програмного забезпечення включає декілька основних етапів [1]: передпроектна підготовка; специфікація вимог до системи; проектування, розробка і тестування програмного забезпечення та впровадження системи (рис.1). Розробка програмного забезпечення має справу з проблемами якості, вартості та надійності. Складність програмного забезпечення порівнянна зі складністю сучасних машин. Саме це вимагає від розробника постійного контролю якості коду, що здійснюється методом тестування програмного забезпечення, яке використовується для виміру якості додатку, який розробляється. Виходячи з аналізу основних завдань розробки програмного забезпечення [1-3], можна зробити висновок, що участь людини повинна бути найменшою в процесі вирішення цих завдань.

Рисунок 1 – Основні етапи розробки додатку

До інших задач можна віднести: знаходження методів, які дозволять зробити процес розгортання додатків простішим; зменшення витрат на конфігурування та підтримку серверів і мережевого обладнання; збільшення доступності додатків. Та, базуючись на обраних рішеннях і проведених дослідженнях, реалізувати систему автоматичного тестування та розгортання мережних сервісів, виконати тестування правильності конфігураційних параметрів, використовуючи її запуск. Проаналізувавши сучасні середовища розробки програмних продуктів маємо сервіси, які виконують тестування та автоматичне розгортання проекту [2-4, 5] - сервіси безперервної інтеграції. В роботі кожного з них є чіткий алгоритм роботи системи, який дозволяє автоматизувати про-

цес автоматичного тестування та розгортання мережевого сервісу та її архітектури (рис. 2).

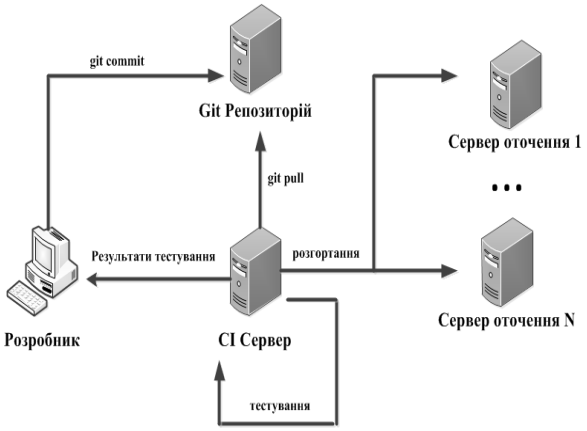

Рисунок 2 – Архітектура системи

Відсутність поєднань готових рішень, які дозволяють розгортати великі системи, призводить до написання подібних скриптів великою кількістю розробників програмних продуктів [1-4]. Тому прийнято рішення щодо поєднання систем безперервної інтеграції з іншими сервісами, призначеними для спрощення процесу розгортання, а саме систем автоматичного конфігурування.

Технології проектування

При розробці програм важливим є використання системи контролю версій. В якості такої системи обрано Git, як провідної серед розробників [6]. Проаналізувавши сучасні тенденції розгортання додатків зроблено висновок, що найчастіше для цих цілей використовуються хмарні технології [7-8], які зменшують витрати на конфігурацію та підтримку працездатності серверів і мережевого обладнання, збільшуючи доступність додатків за рахунок готових рішень провайдерів [9]. Актуальність обумовлюється і тим, що в Україні вже існують проекти переходу на хмарні технології IT-інфраструктур державних органів. Для отримання обчислювальної потужності при розгортанні проекту обираємо провайдера послуг хмарного хостингу або віртуального виділеного серверу.

Підприємства активно впроваджують SaaS, використовуючи хостинги та послуги при зберіганні інформації. Доставка SaaS-додатків через хмару надає їм можливості звільнитися від проблем конфігурування [9]. Після аналізу хмарних технологій [8] та сервісів, обрано Amazon Web Services. Перехід до безперервної інтеграції знижує трудомісткість інтеграції і робить її передбачувано при усуненні помилок. Для розгортання та тестування проекту обрано CircleCI, який інтегрується до системи контролю версій, підтримує налагодження неправильного розгортання. Конфігурація розгортання і тестування знаходиться в проекті. В ролі системи керування конфігураціями обрано Ansible, яка переконфігуровує систему додаванням лише декількох нових рядків у сценарій. Оскільки при тестуванні та налагодженні скриптів систем керування конфігураціями використання хмари є дорогим, а пересилка даних займає багато часу, вирішено, для оцінювання правильності конфігурування, ці процеси проводити локально. Для цього використовуємо Vagrant, який на базі провайдерів віртуальних машин створює кластери на локальній машині і дозволяє конфігурувати кластер за допомогою вбудованого модулю provisioner з підтримкою Ansible [10].

Реалізація системи

Перед початком роботи створюємо обліковий запис AWS. Після реєстрації, для конфігурування сервісів за допомогою веб-інтерфейсу, маємо доступ до консолі менеджменту. Оскільки для розгортання додатку потрібні комп'ютери, конфігуруємо доступ до них. Для реалізації безпеки створюємо ключі та виконуємо налаштування ролей доступу до портів і конфігуруємо спільні ресурси. Спираючись на розроблену модель системи (рис. 3), наводимо етапи її реалізації

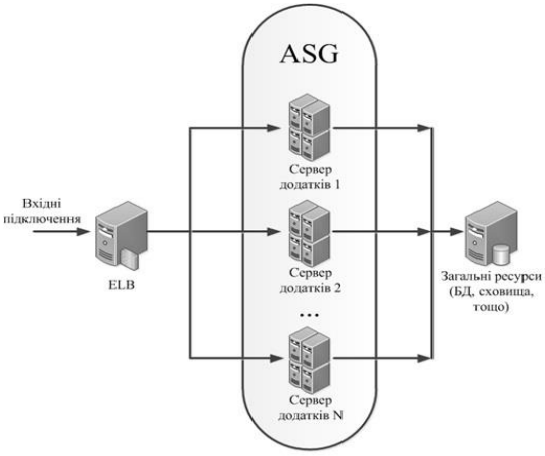

Рисунок 3 – Модель системи

Виходячи з того, що розгортання проекту виконується автоматично, створюємо ролі для доступу до сервісів Amazon - Identity and Access Management і надаємо права доступу новому акаунту. До акаунту генеруються відповідні повноваження автентифікації, які складаються з ідентифікатору доступу і секретного паролю. Виходячи із завдання, використовуємо сервіси надання віртуальних обчислювальних станцій Elastic Compute Cloud (EC2) для створення кластеру, автоматичного балансування навантаження (Elastic Load Balancing) при розподіленні навантаження між машинами в кластері, для аналізу метрик віртуа-

льних машин (CloudWatch), а також автоматичного горизонтального масштабування машин, яке полягає у зміні кількості доступних серверів Auto Scaling. Окремо використовуємо сервіс створення та збереження знімків машин Amazon Machine Image (AMI).

Сама конфігурація для даної системи безперервного розгортання знаходиться безпосередньо у кореневому каталозі репозиторію проекту у файліі circle.yml. Базова конфігурація для такої системи наведена у лістингу 1.

Лістинг 1 – Конфігурація circle.yml

```
machine:
  environment:
    ENV_VARIABLE: test
checkout:
  post:
    - git submodule add git@deployment-repo.git deployment
test:
  override:
    - ./perform_test.sh
deployment:
  production:
    branch: master
    commands:
      - cd deployment; sudo ./deploy.sh
```

Дана конфігурація показує, що у машини нема додаткових налаштувань. Скрипт, який виконує тестування розміщено у файлі perform_test.sh, зміст якого залежить від самого проекту. Оскільки розгортання проекту виконується за допомогою скриптів з іншого репозиторію, то його додано як підмодуль даної програми і він клонується у директорію deployment, яка створена у самому проекті. Налаштування для зв'язку з проектом конфігуруємо з веб-консолі CircleCI. Для виконання розгортання системи на кластері Amazon Web Services використовується скрипт, лістинг якого наведено у deploy.sh (лістинг 2).

Лістинг 2 – Розгортання системи

```
#!/usr/bin/env bash
prepare_key(){ chmod 0400 files/Test.pem }
add_ppa() {
  grep -h "^deb.*$1" /etc/apt/sources.list.d/* > /dev/null 2>&1
  if [ $? -ne 0 ]
  then
    echo "Adding ppa:$1"
    sudo add-apt-repository -y ppa:$1
    return 0
  fi
  echo "ppa:$1 already exists"
  return 1
}
install_ansible() {
  add_ppa ansible/ansible
  apt-get update
  apt-get install ansible -y --force-yes
}
prepare_localhost() { ansible-playbook prepare_localhost.yml }
deploy_cluster() { ansible-playbook deploy_cluster.yml --private-key=files/Test.pem }
main() {
  install_ansible
  prepare_localhost
  prepare_key
  deploy_cluster
}
main
```

Функція main викликається після ініціалізації та складається із послідовності дій. Перша - виконує установку Ansible, друга – підготовку локальної машини за допомогою вже налаштованої системи, третя - змінює права доступу до ключа, який використовується при зв'язку з серверами кластеру.

На етапі підготовки локального серверу встановлюється необхідне програмне забезпечення для роботи з AWS, розширюється набір функцій Ansible для полегшення розгортання кластеру, а також модифікуються права доступу до AWS та ключі для машин, які були створені на попередньому етапі роботи.

Конфігураційні скрипти Ansible розміщуються у файлах, які написані на мові розмітки YAML та мають назву – Playbooks, послідовності дій, які створюють кластер.

На наступному етапі створюється сервер з параметрами, які відповідають завданню з ключем, який створено на попередньому етапі, а також отримується адреса сервера для подальшого використання. Подальшим кроком є розгортання проекту на сервері - деталі реалізації залежать від поставлених завдань. Для розгортання можна використовувати певні сервіси [9,10], це дозволяє уникнути конфігурування Ansible. Підхід, який запропоновано в роботі, дозволяє більш гнучко підходити до задач розгортання додатків, так як система складається з екосистеми сервісів, що розташовані в різних репозиторіях. Розгортання системи виконується за допомогою включення playbook з назвою deploy.yml.

На наступному етапі створюємо образ машини (Amazon Machine Image), з якої складається кластер і конфігурації для запуску автоматичного масштабування (Auto Scaling). Після цього створимо або модифікуємо існуючий балансувальник навантаження (Elastic Load Balancer), який виконує розподілення запитів між усіма серверами кластеру та слідкує, які з машин доступні у даний момент часу. Далі здійснюється заміна існуючого кластеру або створюється новий з конфігурацією

автоматичного масштабування, а також створюються конфігурації, які змінюють кількість серверів у кластері, наприклад на основі високого навантаження протягом заданого часу. Потім очищення старих образів серверів і конфігурацій запуску кластерів та сервер, який використовувався для створення образу, зупиняється.

Оскільки процес тестування скриптів займає тривалий час, то їх краще тестувати на локальній машині. Для цих цілей використовуємо Vagrant, який створює задану кількість віртуальних серверів. Конфігурація наведена у лістингу 3 і знаходиться у файлі Vagrantfile.

Лістинг 3 – Конфігурація Vagrant

```
# -*- mode: ruby -*-
# vi: set ft=ruby :
Vagrant.configure(2) do |config|
  config.vm.box = "ubuntu/trusty64"
  config.vm.network "private_network", ip: "192.168.33.10"
  config.vm.provider "virtualbox" do |vb|
    vb.memory = "1024"
  end
  config.vm.provision "ansible" do |ansible|
    ansible.playbook = "deploy.yml"
    ansible.groups = {
      "vagrant" => ["default"]
    }
  end
end
```

В такій конфігурації створюється віртуальна машина з версією Ubuntu, у якій емульовано адресу інтерфейсу з приватною адресою. На машині також виконується конфігурування апаратних засобів - кількість доступної оперативної пам'яті чи процесорів і максимальний коефіцієнт їх використання. Для виконання запуску віртуального серверу використовуються команда vagrant up, яка виконує запуск машини та скрипти Ansible. Для виконання запуску конфігурування при ввімкненому сервері використовуються команда vagrant provision.

Після правильного конфігурування можна виконувати розгортання кластеру у середовищі віртуальної приватної хмари. Сам скрипт deploy.sh виконує автоматичне конфігурування кластеру, це дозволяє зробити процес автоматизованим.

Тестування роботи системи

Після успішного конфігурування системи перевіряємо її працездатність. Для цього створюємо новий репозиторій, у якому необхідно додати circle.yml із попереднього етапу досліджень. При необхідності його можна замінити іншим. Якщо скрипти для розгортання проекту знаходяться в іншому репозиторії, оскільки вони спільні для усієї системи, необхідно додати і його.

Після проведення коміту до нового репозиторію з основним проектом у консолі AWS можна відслідкувати те, як один за одним виконуються кроки розгортання кластеру в потрібному регіоні. Спочатку створюється новий сервер, який використовується для конфігурування кластеру та вимикається наприкінці роботи (рис. 4).

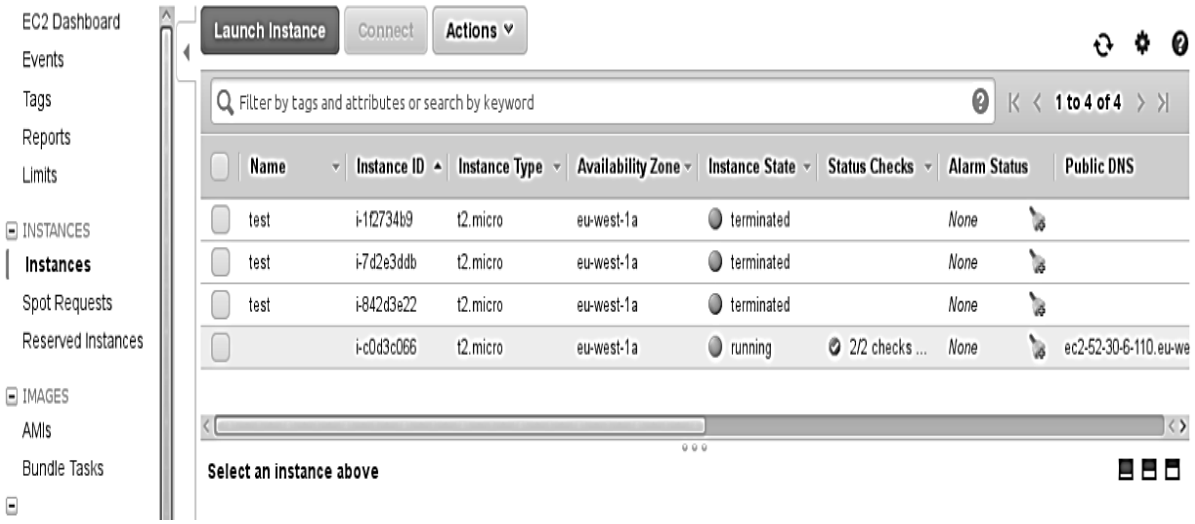

Рисунок 4 – Список активних серверів EC2 потрібного регіону

Далі спостерігаємо, як автоматично створюється конфігурація для запуску нових серверів (рис. 5), а також створюються самі правила для масштабування, разом з автоматичним балансувальником навантаження (рис. 6).

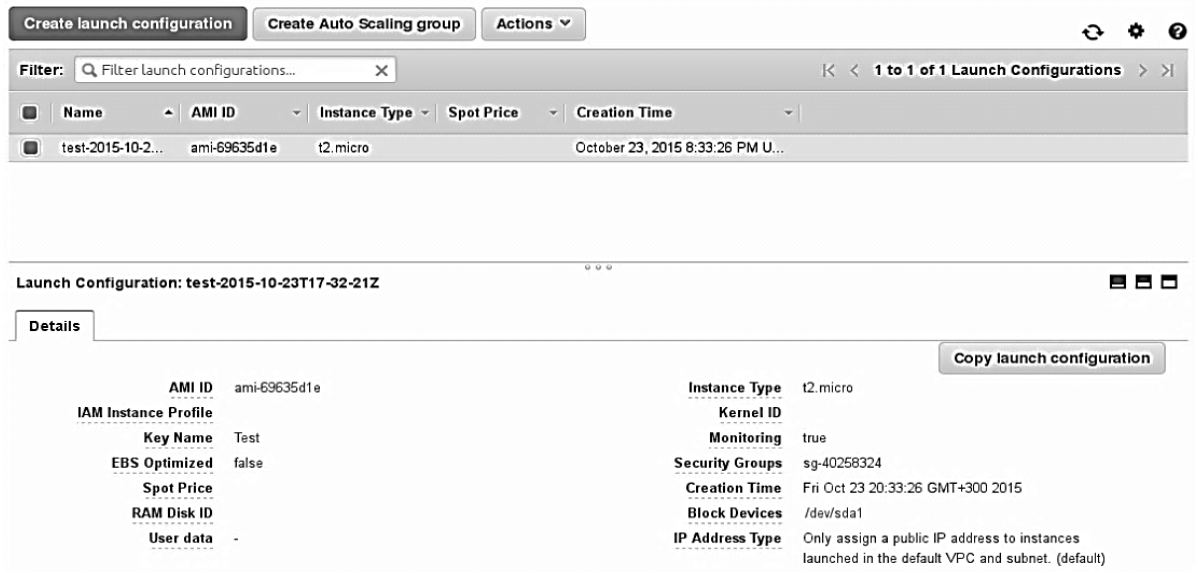

Рисунок 5 – Автоматично створена конфігурація запуску сервера

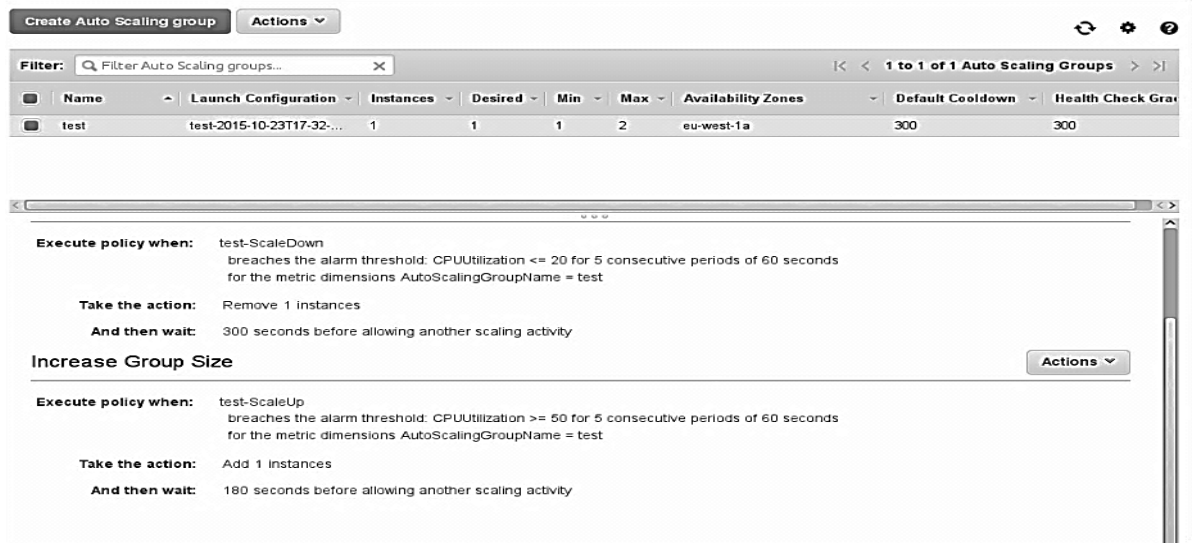

Рисунок 6 – Конфігурація автоматичного масштабування

Сам процес розгортання кластеру з невеликої кількості машин займає декілька хвилин, а процес конфігурування окремої машини залежить вже від самих конфігураційних скриптів. Вигляд консолі після виконання необхідних дій свідчить про те, що розгортання виявилось успішним. Після розгортання системи можна користуватись усіма перевагами з високої доступності систем та широкими можливостями конфігурування кластеру, використовуючи усі можливості, які надає AWS. Після розгортання додатку виконано тестове розгортання серверів та знайдено середній час кожного розгортання. Результати тестування показали, що розроблена система розгортає нову версію додатку приблизно як і AWS Code Deploy, та в декілька разів перевищує розгортання за допомогою ручного конфігурування системи.

Висновки

В ході роботи виявлено задачі, які виникають при створенні мережного програмного забезпечення та розроблено систему для автоматизації процесу тестування, яка дозволяє на основі обраних критеріїв виконувати розгортання додатків будь-якої складності в середовищі віртуальної приватної хмари. Розроблено конфігурацію для сервісів і проведено її тестування. Наведено метод тестування, з використанням розгортання автоматично масштабованого кластеру на основі мережного додатку, який доводить працездатність системи і може бути використаний в рамках розробки подібних проектів.

Список використаної літератури

1. Build-Deploy-Test. Непрерывная интеграция – [Електронний ресурс] Режим доступу: habra-habr.ru/company/icl_services/blog/262173/.
2. Непрерывная интеграция в процессе гибкой разработки – [Електронний ресурс] Режим доступу: www.ibm.com/developerworks/ru/library/r-continuous-integration-agile-development/.
3. Киричек Г.Г. Модель оцінки плагіату програмного коду на основі системи контролю версій / Г.Г. Киричек, О.О. Киричек // Східно-європейський журнал передових технологій. – №2/2(56). – 2012. – С. 25-28.
4. TeamCity – [Електронний ресурс] Режим доступу:jetbrains.ru/products/teamcity/.
5. Система управления конфигурацией Ansible – [Електронний ресурс] Режим доступу: blog.selectel.ru/sistema-upravleniya-konfiguraciej-ansible.
6. The Simplest Way to Build and Deploy Web Applications to the Cloud? – [Електронний ресурс] Режим доступу:www.ctl.io/blog/post/the-simplest-way-to-build-and-deploy-web-applications-to-the-cloud.
7. Five requirements for deploying an application in a public cloud – [Електронний ресурс] Режим доступу: searchcloudcomputing.techtarget.com/tip/Five-requirements-for-deploying-an-application-in-a-public-cloud.
8. Amazon Web Services – [Електронний ресурс] Режим доступу: aws.amazon.com/ru/.
9. Облачные решения (SaaS) – [Електронний ресурс] Режим доступу: store.softline.ru/saas/.
10. Хамбл Д. Непрерывное развертывание ПО / Д. Хамбл, Д. Фарли. – М.: Вильямс, 2011. – С.432.

Надійшла до редакції 21.03.2016

Г.Г. КИРИЧЕК, Р.О. КОТОВ
Запорожский национальный технический университет

СИСТЕМА АВТОМАТИЧЕСКОГО ТЕСТИРОВАНИЯ И РАЗВЕРТЫВАНИЯ СЕТЕВЫХ СЕРВИСОВ

В работе проводится анализ методов развертывания сетевых приложений и предлагаются решения, которые нацелены на сокращение времени и повышение качества развертывания сетевых сервисов в среде виртуального частного облака, с использованием автоматического тестирования исходного кода.

Ключевые слова: облачные технологии, непрерывная интеграция, кластер, балансировка нагрузки.

G.G. KIRICHEK, R.O. KOTOV
Zaporizhzhya National Technical University

SYSTEM OF AUTOMATIC TESTING AND DEPLOYMENT OF NETWORK SERVICES

The article analyzes the methods of deploying network applications and offers the solutions that aim to reduce the time and improve the quality of the deployment of network services in environment of virtual private cloud, using automated testing source code. The purpose is to develop a system for the automatic deployment of network applications after confirming the successful execution of test operations, as the result of which decreases the time to develop an able-bodied version in environment of the virtual private cloud. Based on the analysis of the main objectives of develop, the conclusion is that human involvement should be minimal in the process of solving these problems. Based on studies conducted we decided to implement an automated testing and deployment of the network services is testing the parameters of configuration using its launch. The transition to continuous integration reduces integration complexity and makes it predictable while eliminating errors. To deploy and test the project CircleCI is selected, which is integrated to the version control system that supports debugging of improper deployment. The configuration of deployment and testing is in the project. Based on fact that the deployment of project is done automatically, roles for access to services Amazon - Identity and Access Management are created and permissions for new account are provided. Account authentication is generated by the appropriate authority, which consist of access ID and secret password. The work revealed the problems which arise when creating network software. A system to automate the testing process was developed, which allows done of applications of any complexity in the environment of virtual private cloud.

Keywords: cloud technology, continuous integration, cluster, load balancing.