REFERENCES

[1] John, G., Mui, A., Vlasceanu, V., 2016. Blue/Green Deployments on AWS (White paper). Amazon Web Services, Inc.

[2] Maxim, T., 2015. Reducing Downtime during Software Deployment. Tampere University of technology.

[3] Vincent, E., Brandic, I., Maurer, M., Breskovic, I., n.d. SLA-Aware Application Deployment and Resource Allocation in Clouds. Presented at the 2011 35th IEEE Annual Computer Software and Applications Conference Workshops, IEEE

[4] Cheng, F.-T., Wu, S.-L., Tsai, P.-Y., Chung, Y.-T., Haw-Ching, Y., 2005. Application Cluster Service Scheme for Near-Zero-Downtime Services. Presented at the Robotics and Automation, 2005. ICRA 2005. Proceedings of the 2005 IEEE International Conference, IEEE. doi:10.1109/ROBOT.2005.1570743

[5] Sun, R., Yang, J., He, Z., 2013. An Approach to Minimizing Downtime Induced by Taking Live Snapshot of Virtual Cluster. Presented at the 2013 International Conference on Cloud and Service Computing, IEEE. doi:10.1109/CSC.2013.18

[6] Cornell, D., LarsFeldmann, L., Afanasjev, A., Dimitrov, E., 2005. Method and Apparatus for Facilitating Deployment of Software Applications with Minimum System Downtime. US 20050257216A1.

[7] Rajamani, K., Viswanathan, G., Li, W., Iyer, C., 2005. Minimizing Downtime for Application Changes in Database Systems. US 2005/0251523 A1.

[8] Ponemon Institute, 2016. Cost of Data Center Outages. Available at: http://datacenterfrontier.com/white-paper/cost-data-center-outages/.

[9] Jez Humble, D.F., 2010. Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation 1st Edition. Pearson Education, Inc.

[10] M. Miglierina, “Application Deployment and Management in the Cloud,” presented at the 16th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing, 2014.

[11] Apache Tomcat® 8.0.x 2017, Computer Program, Apache Software Foundation, http://tomcat.apache.org/tomcat-8.0-doc/.

[12] D. Rajdeep, R. A Reddy, and K. Dharmesh, “Virtualization vs Containerization to Support PaaS,” in *Cloud Engineering (IC2E), 2014 IEEE International Conference on*, Boston, MA, USA, 2014.

[13] R. David K, *Kubernetes - Scheduling the Future at Cloud Scale*. CA USA: O’Reilly and Associates
.

[14] C. Damodar, *Tomcat 6 Developer’s Guide*. MUMBAI: Packt Publishing.

[15] "NGINX," *NGINX Wiki,* 2017. [Online]. Available: https://www.nginx.com/resources/wiki/.

[16] V. Moreno, M. R. S, H. E., and L. I. M, "Orchestrating the Deployment of High Availability Services on Multi-zone and Multi-cloud Scenarios," *Journal of Grid Computing*, no. Issue 1/2018, 2017.

[17] W. Timothy, C. Emmanuel, R. K. K., S. Prashant, van der M. Jacobus, and V. Arun, "Disaster recovery as a cloud service: economic benefits & deployment challenges," in *HotCloud'10 Proceedings of the 2nd USENIX conference,* Boston, MA, 2010, pp. 8–8.

[18] X. Wenfeng, Z. Peng, and W. Yonggang, "A Survey on Data Center Networking (DCN): Infrastructure and Operations," *IEEE Communications Surveys & Tutorials*, vol. 19, no. 1, pp. 640–656, Nov. 2016.

[19] L. Eero, I. Juha, and L. Casper, "Problems, causes and solutions when adopting continuous delivery—A systematic literature review," *Information and Software Technology*, vol. Volume 82, pp. 55–79, Feb. 2017.

[20] V. Mateljan, D. Cisic, and D. Ogrizovic, "Cloud Database-as-a-Service (DaaS) - ROI," in *MIPRO, 2010,* Opatija, Croatia, 2010.

[21] H. Ines, M. Marouen, and L. Wajdi, "Cloud Service Delivery across Multiple Cloud Platforms," in *Services Computing (SCC)*, Washington, DC, USA, 2011.

[22] Apache Foundation, "JMeter," *JMeter User's Manual,* Jan-2018. [Online]. Available: http://jmeter.apache.org/usermanual/index.html.

[23] cubicdaiya, walf443, igorastds, and flygoast, *ngx_dynamic_upstream*. 2016. Available: https://github.com/cubicdaiya/ngx_dynamic_upstream.

[24] L. Min-Gu and J. Theresa L, "An Empirical Study of Software Maintenance of a Web-based Java Application," in *21st IEEE International Conference on Software Maintenance*, Budapest, Hungary, 2005.

[25] R. Sebastian and S. Andreas, "The Impact of Context on Continuous Delivery - Full-scale Software Engineering / Current Trends in Release Engineering," RWTH Aachen University, Germany, 2016.

[26] D. Alan, “Software Deployment, Past, Present and Future,” in *Future of Software Engineering, 2007. FOSE ’07,* Minneapolis, MN, USA, 2007.

[27] Gideon and D. Ewa, “Automating Application Deployment in Infrastructure Clouds,” in *Cloud Computing Technology and Science,* Athens, Greece, 2011.

Appendix I - NginX configurations

```
events {
}

http {
   include       mime.types;

         ######python interface for deployment
   server {
      listen       8888;
         # set client body size to 50M #
                  client_max_body_size 50M;
                  proxy_connect_timeout       1000;
                  proxy_send_timeout         1000;
                  proxy_read_timeout         1000;
                  send_timeout               1000;

      location / {
         proxy_pass http://depinterface;
      }

   }

         upstream depinterface {
                  server localhost:5000;
         }

         ######default backends
         upstream backends {
                  zone backends 1m;
                  server 127.0.0.1:8999;
         }

         server {
                  listen 10000;
                  # dynamic upstream config api
                  location /dynamic {
                           allow 127.0.0.1;
                           deny all;
                     dynamic_upstream;
                  }
         }

         server {
                  listen   8080;

      location / {
         # proxy for dynamic upstreams (backends)
```

```
                              proxy_pass http://backends;
        }
            }
}
```

Appendix II - Python helper app source code

```
#app.py

from flask import Flask, request, send_from_directory, jsonify, redirect, url_for
import os
import requests
import docker
import json
from werkzeug.utils import secure_filename
from distutils.dir_util import copy_tree, remove_tree

app = Flask(__name__)



@app.route("/")
def root():
    return send_from_directory('ui', 'index.html')



@app.route("/index")
def index():
    return send_from_directory('ui', 'index.html')



@app.route('/<path:path>')
def send_ui(path):
    return send_from_directory('ui', path)



@app.route('/ui/deploy', methods=['POST'])
def deploy_file_from_ui():
    if request.method == 'POST':
        file = request.files['file']
        if not request.form.get('version'):
            return "version is empty"
        if not request.form.get('appname'):
            return "app package name is empty"
        version = request.form.get('version')
        appname = request.form.get('appname')
        create_workspace(appname, version)
        copy_assets(appname, version)
        filename = secure_filename(file.filename)
        # saving file to workspace
        file.save(os.path.join('workspace/' + appname + '/' + version, filename))
        # create docker file
        create_docker_file(appname, version)
        # get an available port
        port = get_available_port()
        # reserve port
        reserve_app_port(appname, version, port)
        # build image
        build_docker_image(appname, version)
        # clean workspace
        clean_assets(appname, version)
        run_container(appname, version, port)
        add_to_proxy(port)
        return redirect(url_for('index'))


@app.route('/api/activate', methods=['GET'])
def activate():
    if request.method == 'GET':
        if not request.args.get('version'):
            return "version is empty"
        if not request.args.get('appname'):
            return "app package name is empty"
        version = request.args.get('version')
        appname = request.args.get('appname')

        # start and connect app to proxy
        port = get_port_for_stopped_app(appname, version)['port']
        if not is_running(appname, version):
            print("Starting application:" + appname + ":" + version)
            run_container(appname, version, port)
        up("127.0.0.1:" + port)
        # down other applications
        applications = get_all_apps()
        print("Disconnecting other applications")
        for application in applications:
            if (application['appname'] != appname) or (application['version'] != version):
                down("127.0.0.1:" + application['port'])
        return "activated " + appname + " " + version


@app.route('/api/backends')
def backends():
    return jsonify(get_all_apps())


@app.route('/api/apps/<appname>/<version>/status')
def get_app_status(appname, version):
    if not request.args.get('health_route'):
        return "health_route is empty"
    health_route = request.args.get('health_route')
    port = get_port_for_stopped_app(appname, version)['port']
    # send a get request to the app's health route, timeout for 3 seconds
    try:
        r = requests.get('http://127.0.0.1:' + port + health_route, verify=False, timeout=3)
    except requests.exceptions.ConnectionError:
        return jsonify({'status': 'down', 'code': 500, 'err': 'Connection refused'})
    if r.status_code == 200:
        return jsonify({'status': 'up', 'code': r.status_code})
    else:
        return jsonify({'status': 'down', 'code': r.status_code})


# retrieve all available apps
def get_all_apps():
    apps = []
    client = docker.from_env()
    images = client.images.list(all=True)
    j = 0
    while j < len(images):
        image = images[j]
        if len(image.tags):
            for tag in image.tags:
                tag_parts = tag.split(':')
                appname = tag_parts[0]
                version = tag_parts[1]
                if appname != 'ubuntu':
                    running = is_running(tag_parts[0], tag_parts[1])
                    connected = is_connected(tag_parts[0], tag_parts[1])
                    apps.append({"appname": appname,
                                 "version": version,
                                 "port": get_port_for_stopped_app(appname, version)['port'],
                                 "isConnected": connected,
                                 "isRunning": running})
        j = j + 1
    return apps
```

```
def is_connected(appname, version):
    proxy_backends = get_all_proxy_connections()
    for backend in proxy_backends:
        if backend['appname'] == appname and backend['version'] == version:
            return backend['isConnected']
    return False


def does_connection_exists(port):
    r = requests.get('http://127.0.0.1:10000/dynamic?upstream=backends&verbose=')
    lines = r.text.split('\n')
    i = 0
    while i < len(lines):
        if len(lines[i]) > 0:
            line_parts = lines[i].split(' ')
            backend_parts = line_parts[1].split(':')
            if port == backend_parts[1]:
                return True
        i = i + 1
    return False


# fetch backends from nginx proxy
def get_all_proxy_connections():
    r = requests.get('http://127.0.0.1:10000/dynamic?upstream=backends&verbose=')
    lines = r.text.split('\n')
    i = 0
    backends = []
    while i < len(lines):
        if len(lines[i]) > 0:
            line_parts = lines[i].split(' ')
            backend_parts = line_parts[1].split(':')
            port = backend_parts[1]
            status = line_parts[len(line_parts) - 1]
            if status == "down;":
                connected = False
            else:
                connected = True
            app_details = get_app_for_port(port)
            backends.append({"appname": app_details.get('appname'),
                             "version": app_details.get('version'),
                             "port": port,
                             "isConnected": connected})
        i = i + 1
    return backends


@app.route('/api/backends/<application>/down')
def down(application):
    r = requests.get('http://127.0.0.1:10000/dynamic?upstream=backends&server=' + application + '&down=')
    return r.text


@app.route('/api/backends/<application>/up')
def up(application):
    port = application.split(':')[1]
    if does_connection_exists(port):
        r = requests.get('http://127.0.0.1:10000/dynamic?upstream=backends&server=' + application + '&up=')
    else:
        r = requests.get('http://127.0.0.1:10000/dynamic?upstream=backends&server=' + application + '&add=')
    return r.text


@app.route('/api/backends/<application>/add')
def add(application):
    r = requests.get('http://127.0.0.1:10000/dynamic?upstream=backends&server=' + application + '&add=')
    return r.text


@app.route('/api/backends/<application>/remove')
def remove(application):
    appname = request.args.get('appname')
    version = request.args.get('version')
    terminate_container(appname, version)
    r = requests.get('http://127.0.0.1:10000/dynamic?upstream=backends&server=' + application + '&remove=')
    terminate_image(appname, version)
    release_app_port(appname, version, application.split(':')[1])
    return r.text


@app.route('/api/backends/stop')
def stop():
    appname = request.args.get('appname')
    version = request.args.get('version')
    terminate_container(appname, version)
    return "container stopped and removed"


@app.route('/api/backends/start')
def start():
    appname = request.args.get('appname')
    version = request.args.get('version')
    if not is_running(appname, version):
        port = get_port_for_stopped_app(appname, version)['port']
        run_container(appname, version, port)
    return "container started"


@app.route('/api/test')
def test():
    var = is_running("myapp", "1.0.0")
    return "test"


@app.route('/api/deploy', methods=['POST'])
def deploy_file():
    if request.method == 'POST':
        file = request.files['file']
        if not request.args.get('version'):
            return "version is empty"
        if not request.args.get('name'):
            return "app name is empty"
        version = request.args.get('version')
        appname = request.args.get('name')
        create_workspace(appname, version)
        copy_assets(appname, version)
        filename = secure_filename(file.filename)
        # saving file to workspace
        file.save(os.path.join('workspace/' + appname + '/' + version, filename))
        # create docker file
        create_docker_file(appname, version)
        # get an available port
        port = get_available_port()
        # reserve port
        reserve_app_port(appname, version, port)
        # build image
        build_docker_image(appname, version)
        # clean workspace
        clean_assets(appname, version)
        run_container(appname, version, port)
        add_to_proxy(port)
        return 'file deployed successfully'


def add_to_proxy(port):
    requests.get('http://127.0.0.1:10000/dynamic?upstream=backends&server=127.0.0.1:'       +
str(port) + '&add=')
    requests.get('http://127.0.0.1:10000/dynamic?upstream=backends&server=127.0.0.1:'       +
str(port) + '&down=')


def run_container(appname, version, port):
    client = docker.from_env()
    client.containers.run(appname + ':' + version, 'tomcat/bin/catalina.sh run', detach=True,
                          ports={'8080/tcp': port})


def copy_assets(appname, version):
    copy_tree('assets/jre', 'workspace/' + appname + '/' + version + '/jre')
    copy_tree('assets/tomcat', 'workspace/' + appname + '/' + version + '/tomcat')


def clean_assets(appname, version):
    remove_tree('workspace/' + appname + '/' + version + '/tomcat')
    remove_tree('workspace/' + appname + '/' + version + '/jre')


def build_docker_image(appname, version):
    client = docker.from_env()
    client.images.build(path='workspace/' + appname + '/' + version,
                        tag=appname + ':' + version,
                        rm=True)


def terminate_container(appname, version):
    client = docker.from_env()
    containers = client.containers.list(all=True)
    i = 0
    while i < len(containers):
        tag = containers[i].attrs['Config']['Image']
        tag_parts = tag.split(":")
        if appname == tag_parts[0] and version == tag_parts[1]:
            # stop container
            containers[i].kill()
            containers[i].remove()
            print("terminated " + appname + ":" + version)
            break
        i = i + 1


def terminate_image(appname, version):
    client = docker.from_env()
    images = client.images.list(all=True)
    i = 0
    while i < len(images):
        if len(images[i].tags) > 0:
            for tag in images[i].tags:
                tag_parts = tag.split(":")
                if appname == tag_parts[0] and version == tag_parts[1]:
                    # remove image
                    client.images.remove(image=appname + ':' + version, force=True)
                    print("terminated image " + appname + ":" + version)
                    break
        i = i + 1


def create_docker_file(appname, version):
    # port = get_available_port()
    f = open('workspace/' + appname + '/' + version + '/Dockerfile', 'w')
    f.write('# Tomcat 8 customized\n')
    f.write('#')
    f.write('# VERSION               0.0.1\n')
    f.write('\n')
    f.write('FROM      ubuntu\n')
    f.write('LABEL Description="This image is used to run a customized tomcat server" Version="1.0"\n')
    f.write('ADD tomcat /tomcat\n')
    f.write('ADD jre /jre\n')
    f.write('ADD *.war /tomcat/webapps/\n')
    f.write('ENV JAVA_HOME /jre\n')
    f.write('#ENV JAVA_OPTS -Dport.shutdown=8065 -Dport.http=8060\n')
#       f.write('#RUN sed "s/8080/' + str(port) + '/g" < /tomcat/conf/server.xml > /tmp/server.xml\n')
#    f.write('#RUN cp /tmp/server.xml /tomcat/conf/server.xml\n')
```

```
    f.write('EXPOSE 8080')
    f.close()


def reserve_app_port(appname, version, port):
    f = open('conf/app_ports', 'a')
    f.write(str(appname) + ',' + str(version) + ',' + str(port) + '\n')
    f.close()


def release_app_port(appname, version, port):
    content_to_write = ''
    with open('conf/app_ports') as f:
        content = f.read().splitlines()
    i = 0
    while i < len(content):
        parts = content[i].split(',')
        if not (parts[0] == appname and parts[1] == version and parts[2] == str(port)):
            content_to_write += content[i] + '\n'
        i = i + 1
    f = open('conf/app_ports', 'w')
    f.write(content_to_write)
    f.close()


def is_port_taken(port):
    with open('conf/app_ports') as f:
        content = f.read().splitlines()
    i = 0
    while i < len(content):
        parts = content[i].split(',')
        if parts[2] == str(port):
            return True
        i = i + 1
    return False


def get_app_for_port(port):
    with open('conf/app_ports') as f:
        content = f.read().splitlines()
    i = 0
    while i < len(content):
        parts = content[i].split(',')
        if parts[2] == str(port):
            return {'appname': parts[0], 'version': parts[1], 'port': port}
        i = i + 1
    print("couldn't find app for the port")
    return {'appname': '', 'version': '', 'port': port}


def get_port_for_stopped_app(appname, version):
    with open('conf/app_ports') as f:
        content = f.read().splitlines()
    i = 0
    while i < len(content):
        parts = content[i].split(',')
        if parts[0] == appname and parts[1] == version:
            return {'appname': parts[0], 'version': parts[1], 'port': parts[2]}
        i = i + 1
    print("couldn't find app for the port")
    return {'appname': '', 'version': '', 'port': 0}


def get_available_port():
    p = 8300
    while p < 8500:
        p = p + 1
        if not is_port_taken(p):
            return p
    print("Error! Out of ports")


#   creating workspace for the app and version
def create_workspace(appname, version):
    if not os.path.exists('workspace/' + appname):
        os.makedirs('workspace/' + appname)
    if not os.path.exists('workspace/' + appname + '/' + version):
        os.makedirs('workspace/' + appname + '/' + version)


def is_running(appname, version):
    client = docker.from_env()
    containers = client.containers.list(all=True)
    i = 0
    while i < len(containers):
        tag = containers[i].attrs['Config']['Image']
        tag_parts = tag.split(':')
        if appname == tag_parts[0] and version == tag_parts[1]:
            return True
        i = i + 1
    return False


if __name__ == '__main__':
    app.run(debug=True)
```