Ministry of Higher Education
and Scientific Research
*** * ***
Carthage University
*** * ***
National Institute of Applied Science and Technology

Internship Report

Major: Software Engineering

Level: 3rd Year

Subject:

CI/CD pipeline implementation for Odoo ERP

Made by: **Alaeddine HAMROUN**

Company:

SOFTIFI

Acknowledgement

I have the pleasure of keeping this page as a sign of gratitude and deep appreciation to all those who have helped me directly or indirectly throughout my internship.

I express my deepest thanks to **Ms. Mouna Jaballah**, my professional supervisor and DevOps engineer at SOFTIFI, for her guidance throughout this internship, her wise advice and her encouragement.

Summary

Figure Table:

I. Introduction

During 1 Month (01July – 31 July), I have done an internship at SOFTIFI.
In this internship, I was able to discover the professional life and the working conditions within a company. Also, I was able to develop my skills and learn more about developing and operating an application.

My internship supervisor is Ms. Mouna Jaballah, Thanks to her consideration, guidance and quality of support I was able to learn in excellent conditions and enjoy the work.

DevOps is an approach based on the synergy between the operational part and the production part. It is within this context, my goal during my internship at SOFTIFI was to set up a CI/CD pipeline for an Odoo ERP application.

II. Enterprise Presentation

SOFTIFI is an IT company that offers a range of Information Technology services designed for business progress. From building ERP systems, outsourcing, to e-commerce solutions, and providing a world-class customer experience through comprehensive digital marketing services. Its goal is to transform creative ideas into practical digital solutions.

The SOFTIFI team work on providing exceptional web and mobile solutions and comprehensive e-marketing services to support our customers' digital transformation.

Their initial focus is on providing a comprehensive suite of managed IT solutions in order to cover clients' requirements.

III. Project Description

Usually, putting an application into production is the final step in a process involving different teams, namely the dev team and the test team. Thus, the development, the test and the release into production are considered as three distinct stages.
Involving so many teams can lead to conflicts since the goal of each team is different from the other. When developers want to innovate and evolve applications, the production team seeks, above all, to maintain the stability of the system. Moreover, everyone follows their own processes, and works with their own tools lacking communication. As a result, the relationships between these teams can be conflicting. These conflicts generate delays in delivery, additional costs for the company and an impact on the satisfaction of the customer which is the focus of the company.
It then becomes obvious that a new approach must be adopted which makes it possible to unify the development and production process in order to avoid all the cited problems previously.
From this, the notion of DevOps was born. It is an approach based on the synergy between the operational part and the production part. The alignment of all teams of the information system on a common objective makes it possible to reduce conflicts between these various stakeholders, to avoid delays due to communication between them, and to improve, therefore, delivery time.
It is within this context, during this internship I aimed to get familiar with DevOps concept and realize a CI/CD pipeline for an Odoo application.

Tasks required in this internship:

- Install Odoo with Docker.
- Create a Docker image for an Odoo application.
- Manage an Azure VM with Ansible.
- Monitor infrastructure with Zabbix.

IV. Internship Progress

After setting up my environment, I have started working on the objectives of the internship. This is what I've learned from my time at SOFTIFI:

1. Odoo ERP:

1.1. What is Odoo ERP?

Enterprise resource planning (ERP) refers to a type of software that organizations use to manage day-to-day business activities such as accounting, procurement, project management, risk management and compliance, and supply chain operations.

Odoo ERP system is enterprise resource planning software used company-wide for the management of business processes. Odoo provides seamlessly integrated functional business apps called Odoo apps that form an ERP solution when combined. Open-source software, Odoo, is available with SaaS subscription pricing as the Enterprise edition or as the Odoo free Community version.

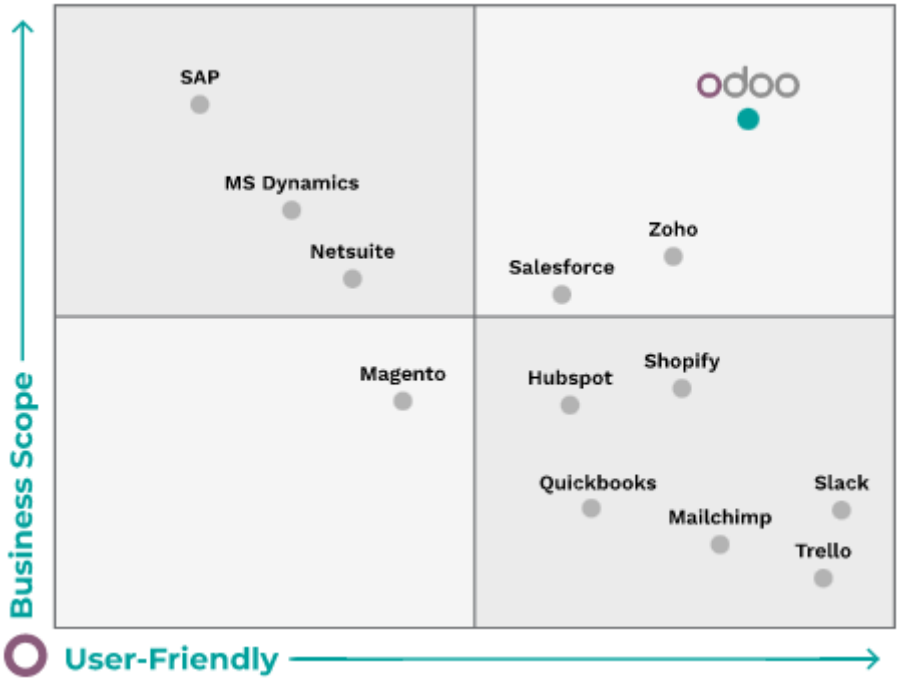


Figure 1: ERP Solutions [1]

I've used the Odoo Community version because it is free of charge and we can implement Odoo on your own server for free.

1.2. Developing Odoo modules

Odoo follows a multitier architecture as follows:

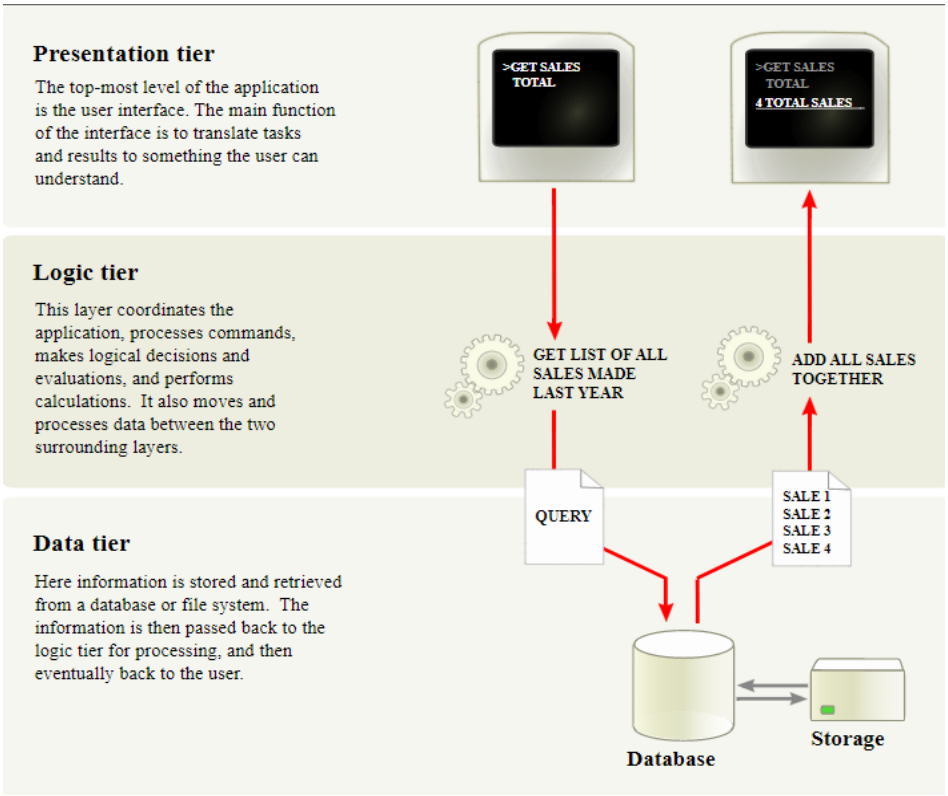


Figure 2: Odoo architecture overview [1]

The presentation tier is a combination of HTML5, JavaScript and CSS. The logic tier is exclusively written in Python, while the data tier only supports PostgreSQL as an RDBMS.

Both server and client extensions are packaged as modules which are optionally loaded in a database. A module is a collection of functions and data that target a single purpose.

Odoo modules can either add brand new business logic to an Odoo system or alter and extend existing business logic.
Modules may also be referred to as addons.
Here is a simplified module structure:

```
module
├── models
│   ├── *.py
│   └── __init__.py
├── data
│   └── *.xml
├── __init__.py
└── __manifest__.py
```

Figure 3: Simplified Module Structure [1]

The bottom line here is that Odoo developers develop and customize modules based on company needs, and that Odoo's server runs on Python and uses PostgreSQL as an RDBMS.

2. Containerization with Docker:

2.1. What is Docker?

Docker is a tool that allows developers, sys-admins etc. to easily deploy their applications in a sandbox (called containers) to run on the host operating system i.e., Linux. The key benefit of Docker is that it allows users to package an application with all of its dependencies into a standardized unit for software development. Unlike virtual machines, containers do not have high overhead and hence enable more efficient usage of the underlying system and resources.

2.2. What is a Docker container?

The industry standard today is to use Virtual Machines (VMs) to run software applications. VMs run applications inside a guest Operating System, which runs on virtual hardware powered by the server's host OS.

VMs are great at providing full process isolation for applications: there are very few ways a problem in the host operating system can affect the software running in the guest operating system, and vice-versa. But this isolation comes at great cost — the computational overhead spent virtualizing hardware for a guest OS to use is substantial.

Containers take a different approach: by leveraging the low-level mechanics of the host operating system, containers provide most of the isolation of virtual machines at a fraction of the computing power.

Containers offer a logical packaging mechanism in which applications can be abstracted from the environment in which they actually run. This decoupling allows container-based applications to be deployed easily and consistently, regardless of whether the target environment is a private data center, the public cloud, or even a developer's personal laptop. This gives developers the ability to create predictable environments that are isolated from the rest of the applications and can be run anywhere.

From an operations standpoint, apart from portability containers also give more granular control over resources giving your infrastructure improved efficiency which can result in better utilization of resources.

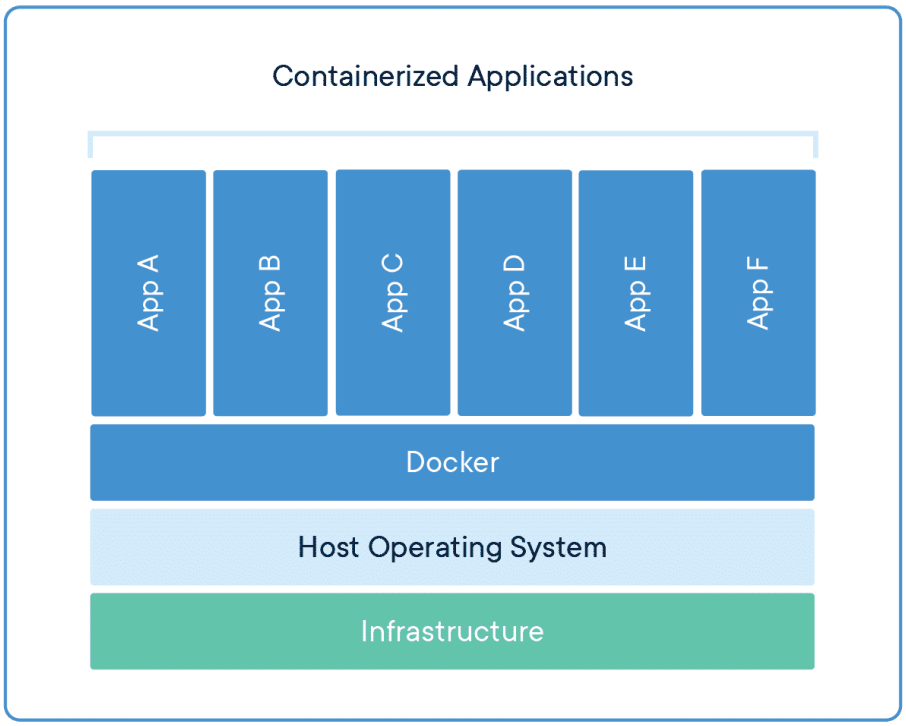


Figure 4: Docker Containers [2]

To build a container, first we need to create a Docker image.
A Docker image is a file used to execute code in a Docker container. Docker images act as a set of instructions to build a Docker container, like a template.

2.3. Using Docker with Odoo

Docker isolates the complexity in developing applications that depends on multiple languages, frameworks, architectures, and discontinuous interfaces between tools in the form of containerized application.

The following figure (figure 5) shows the containers needed to run an Odoo application.

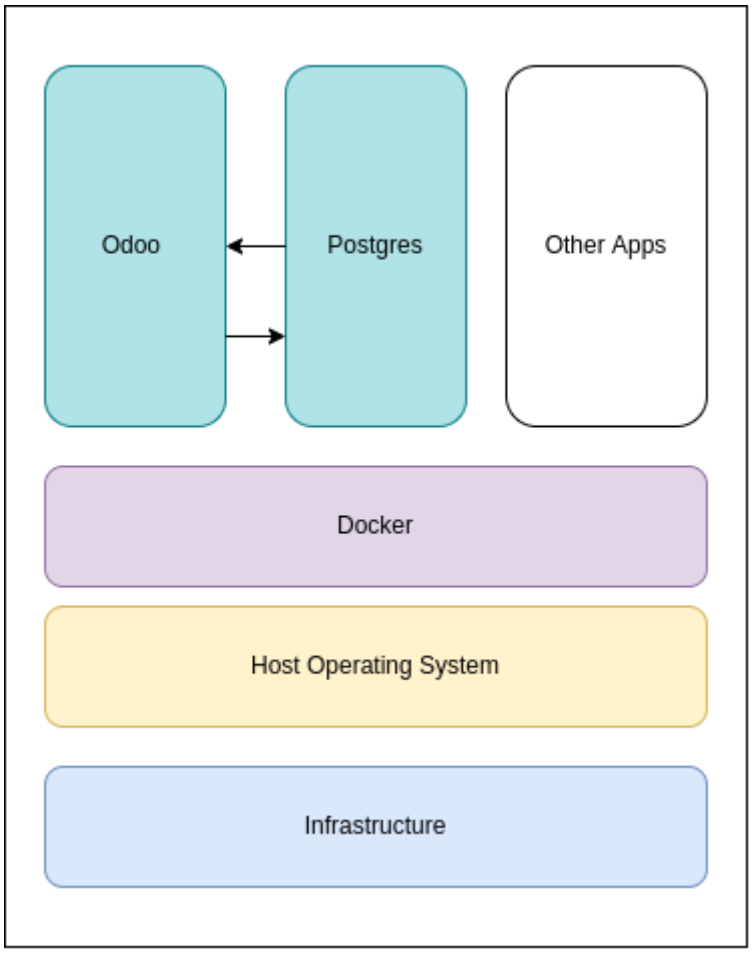


Figure 5: Odoo application on Docker

The official Docker image for Odoo is available in this git repository: [GitHub - odoo/docker]
The repository contains three different versions of Odoo; v13, v14 and v15. I used the version 15 as it is the latest version up to august 2022.
The following figure (figure 6) shows the content of the repository.

```
alaeddine@alaeddine-legion:~/docker$ ls
13.0  14.0  15.0  LICENSE  README.md
alaeddine@alaeddine-legion:~/docker$ ls 15.0/
Dockerfile  entrypoint.sh  odoo.conf  wait-for-psql.py
alaeddine@alaeddine-legion:~/docker$
```

Figure 6: Official Docker image for Odoo

- **Dockerfile**: a simple text file that contains a list of commands that the Docker client calls while creating an image. It's a simple way to automate the image creation process.
- **Entrypoint.sh**: the starting point of Odoo image.
- **Odoo.conf**: Odoo configuration file used by the image.
- Wait-for-psql.py: script used by entrypoint to wait for the postgres server to run so that Odoo can run.

The following command builds the Docker image and tags it with 'alaeddinehamroun/odoo:1.0':

```
docker build -t alaeddinehamroun/odoo:1.0 .
```

- . : meaning from Dockerfile located in current folder.
- -t: to allocate a pseudo-tty.

Next, starting an Odoo instance requires a running PostgreSQL server:
This command creates a psql instance named db:

```
docker run -d -e POSTGRES_USER=odoo -e POSTGRES_PASSWORD=odoo -e
POSTGRES_DB=postgres --name db postgres:13
```

Then we start the instance:

```
docker run -d -p 8069:8069 --name odoo --link db:db -t odoo
```

- -d flag: run this command in detached mode.
- -p flag: creates a firewall rule which maps a container port to a port on the Docker host to the outside world.

By default, this command creates a bridge network.
A bridge network allows containers connected to the same bridge network to communicate, while providing isolation from containers which are not connected to that bridge network.
To inspect the created bridge network, we run the following commands:

```
alaeddine@alaeddine-legion:~/docker/15.0$ docker network ls
NETWORK ID     NAME                   DRIVER    SCOPE
31cc150fd585   bridge                 bridge    local
564a77ebaeca   host                   host      local
1d50ebcaa875   minikube               bridge    local
cb8894157dba   none                   null      local
aef0e671dac9   odoo_default           bridge    local
03dfe6f19645   task-manager_default   bridge    local
alaeddine@alaeddine-legion:~/docker/15.0$ docker inspect 31cc150fd585
[
    {
        "Name": "bridge",
        "Id": "31cc150fd5851317393e702a5441649760fa96f6ac032a4acde178eda1210ea4",
        "Created": "2022-08-10T10:52:44.805598857+01:00",
        "Scope": "local",
        "Driver": "bridge",
        "EnableIPv6": false,
        "IPAM": {
            "Driver": "default",
            "Options": null,
            "Config": [
                {
                    "Subnet": "172.17.0.0/16",
                    "Gateway": "172.17.0.1"
                }
            ]
        },
        "Internal": false,
        "Attachable": false,
        "Ingress": false,
        "ConfigFrom": {
            "Network": ""
        },
        "ConfigOnly": false,
        "Containers": {
            "4fb018bd5a0e1bc89035dbbaf03c69e01463c17fec902b433e03aec681e9859a": {
                "Name": "odoo",
                "EndpointID": "0d3094277f39dc9e5969b835831e24af88b9325e8bd349fddcbc60d53321de2e",
                "MacAddress": "02:42:ac:11:00:03",
                "IPv4Address": "172.17.0.3/16",
                "IPv6Address": ""
            },
            "9481a3375002e1a9e8ebd6364a7fba764bf70839c78303f1dc7b936dabc7df44": {
                "Name": "db",
                "EndpointID": "cf088295de84feda8dd887f894cca069af43ecc74f1e638460c1b195b9d86f1e",
                "MacAddress": "02:42:ac:11:00:02",
                "IPv4Address": "172.17.0.2/16",
                "IPv6Address": ""
            }
        },
        "Options": {
            "com.docker.network.bridge.default_bridge": "true",
            "com.docker.network.bridge.enable_icc": "true",
            "com.docker.network.bridge.enable_ip_masquerade": "true",
            "com.docker.network.bridge.host_binding_ipv4": "0.0.0.0",
            "com.docker.network.bridge.name": "docker0",
            "com.docker.network.driver.mtu": "1500"
        },
        "Labels": {}
    }
]
```

Figure 7: Bridge network inspection

The following figure (figure 8) explains more the port mapping and the created bridge network:

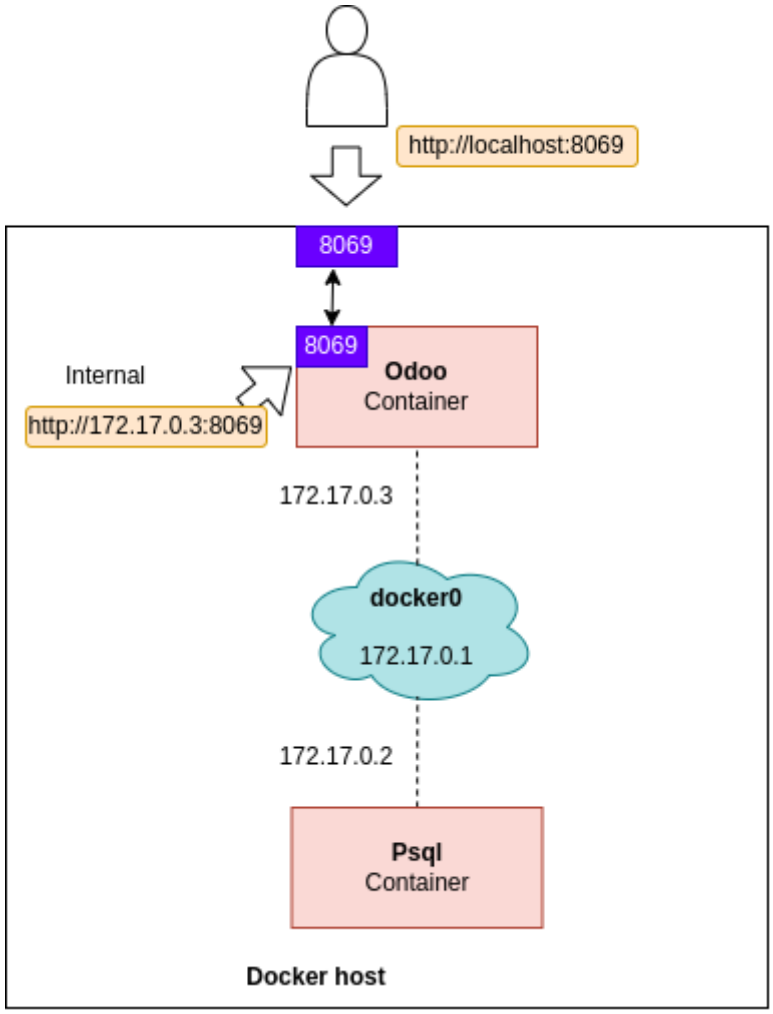


Figure 8: Port mapping and bridge network

One problem I faced when running Odoo application like described above is that every time I stop the container and start it again, my changes are lost. In fact, the Odoo filestore is created inside the container and if the container is removed, the filestore is lost.
To solve this problem, we used Docker volumes.
Volumes are created and managed by Docker and stored in a part of the host filesystem (/var/lib/docker/volumes/ on linux). There are other ways to manage data with Docker but Volumes are the best way to persist data in docker.
When you mount a volume, it may be named or anonymous. Anonymous volumes are not given an explicit name when they are first mounted into a container, so Docker gives them a random name that is guaranteed to be unique within a given Docker host. Besides the name, named and anonymous volumes behave in the same ways.

Beside the volume used to preserve the filestore, we use another named volume to preserve the psql database, another one to mount Odoo addons at /mnt/extra-addons and one more to override the default Odoo configuration file (located at /etc/odoo/odoo.conf) at startup.

The following figure (figure 9) shows the list of used volumes:

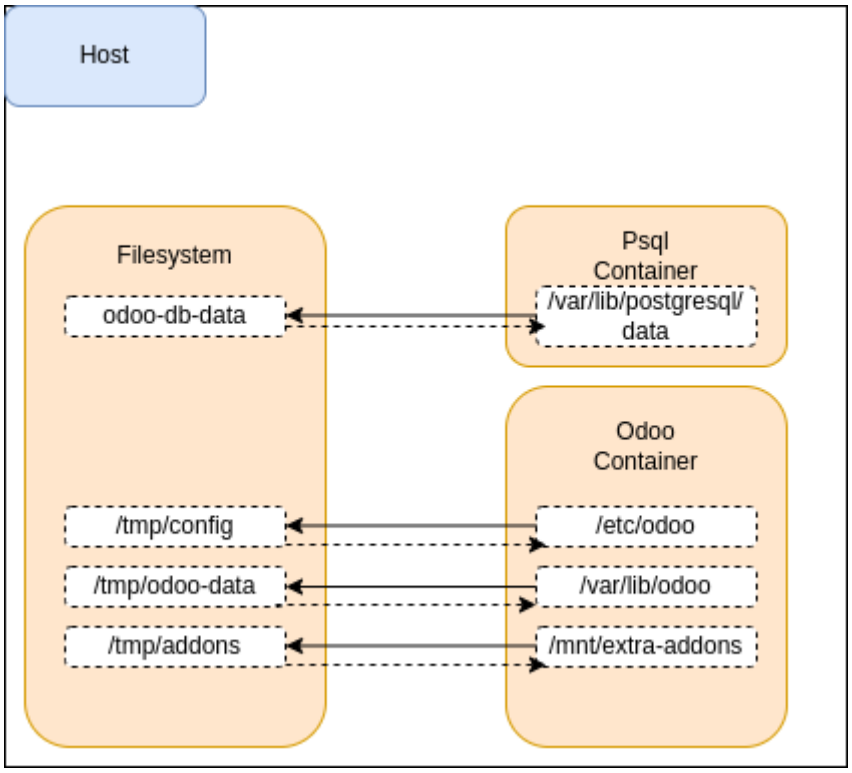


Figure 9: Docker Odoo Volumes

So, the final command would be like this:
For the psql instance:

```
docker run -d -v odoo-db-data:/var/lib/postgresql/data -e
POSTGRES_USER=odoo -e POSTGRES_PASSWORD=odoo -e POSTGRES_DB=postgres -
-name db postgres:13
```

For the Odoo instance:

```
docker run -v /tmp/odoo-data:/var/lib/odoo -v /tmp/addons:/mnt/extra-
addons -v /tmp/config:/etc/odoo --link db:db alaeddinehamroun/odoo:1.0
```

-v flag: to mount volume into container.

One major problem with this solution is that when the PostgreSQL server is restarted, the Odoo instance linked to that server must be restarted as well because the server address has changed and the link is thus broken. Moreover, both of the above commands are pretty long.
So, as suggested from my supervisor, I used Docker compose.
Compose is a tool for defining and running multi-container docker applications. With compose, you use a YAML file to configure your application's services. Then, with a single command, you create and start all the services from your configuration.

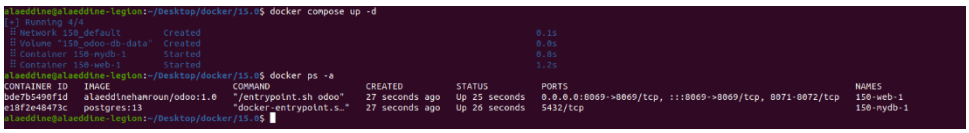
Docker-compose.yaml file:

```yaml
version: '3.1'
services:
  web:
    image: alaeddinehamroun/odoo:1.0
    depends_on:
      - mydb
    ports:
      - "8069:8069"
    volumes:
      - ./addons:/mnt/extra-addons
      - ./config:/etc/odoo
      - odoo-web-data:/var/lib/odoo
    environment:
    - HOST=mydb
    - USER=odoo
    - PASSWORD=myodoo
  mydb:
    image: postgres:13
    volumes:
    - odoo-db-data:/var/lib/postgresql/data
    environment:
      - POSTGRES_DB=postgres
      - POSTGRES_PASSWORD=myodoo
      - POSTGRES_USER=odoo
volumes:
  odoo-web-data:
  odoo-db-data:
```

We add these two lines to the Dockerfile to copy addons inside the container:

```
# Copy entrypoint script and Odoo configuration file
COPY ./entrypoint.sh /
COPY ./odoo.conf /etc/odoo/

# Set permissions and Mount /var/lib/odoo to allow restoring filestore and /mnt/extra-
RUN chown odoo /etc/odoo/odoo.conf \
    && mkdir -p /mnt/extra-addons \
    && chown -R odoo /mnt/extra-addons
VOLUME ["/var/lib/odoo", "/mnt/extra-addons"]

# Expose Odoo services
EXPOSE 8069 8071 8072

# Set the default config file
ENV ODOO_RC /etc/odoo/odoo.conf

COPY wait-for-psql.py /usr/local/bin/wait-for-psql.py
# Copy custom addons (new developed addons)
COPY ./addons /mnt/extra-addons; fi

# Copy extra addons (old-production-ready addons or installed addons from the store)
COPY ./addon_source /opt/addons

# Set default user when running the container
USER odoo

ENTRYPOINT ["/entrypoint.sh"]
CMD ["odoo"]
```

And then we run the command docker-compose up -d.



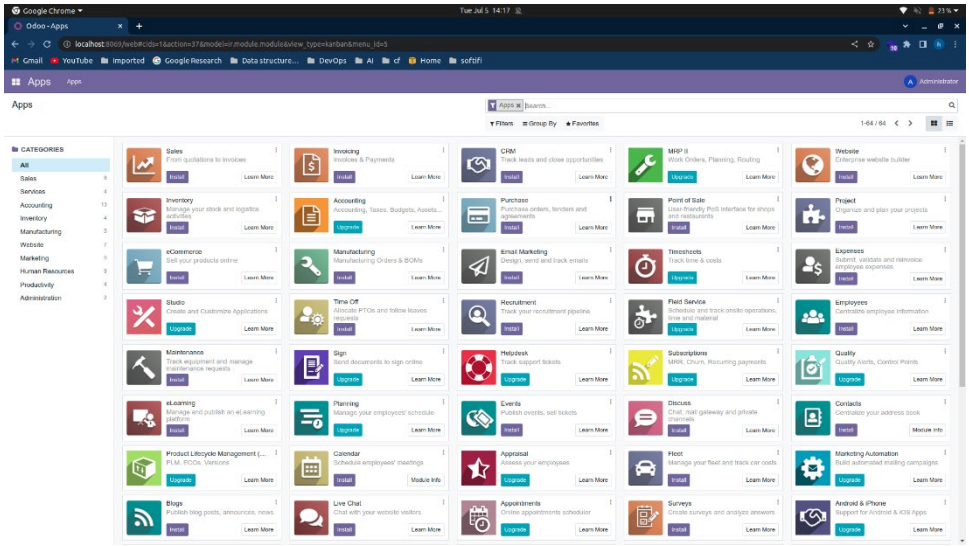


Figure 10: Odoo app

3. CI/CD Pipeline with Jenkins and Bitbucket:

3.1. CI/CD definition

Continuous integration (CI) is the practice of automatically building and unit testing an entire application frequently, ideally on every source code check-in. This concept was first introduced over two decades ago to avoid 'integration hell', which happens when integration is put off till the end of a project.

Continuous delivery (CD) is the practice of deploying every build to a production-like environment and performing automated integration and acceptance testing. It helps reduce the cost, time, and risk of delivering changes by allowing for frequent updates in production.
Sometimes Continuous delivery gets confused with Continuous deployment.
Continuous deployment is the practice of automatically deploying every build to production after it passes its automated tests.

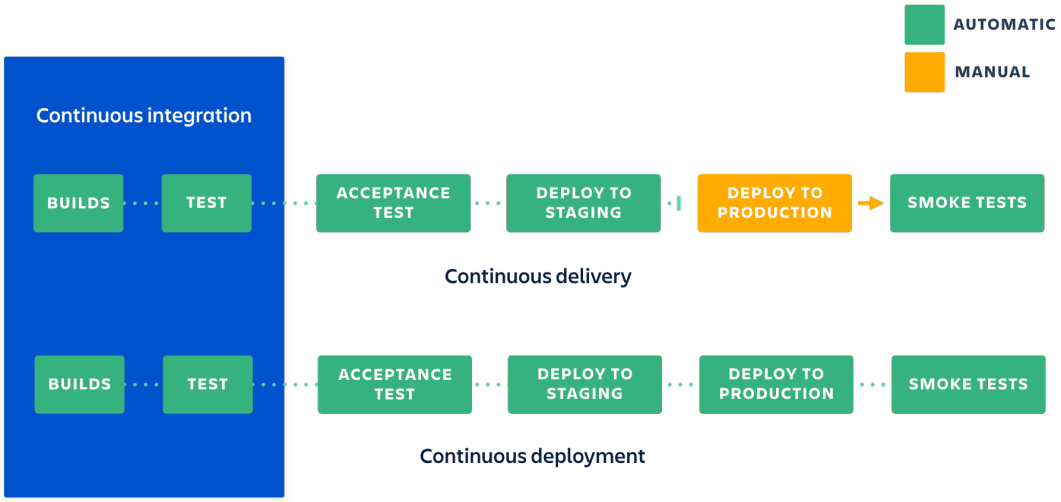


Figure 11: Continuous Delivery vs Continuous Deployment [3]

A build pipeline is the sequence of operations and the tools that perform them between source code and deployed system.
Tool-wise, we start with the source code repository Git. This is where we are going to keep all of our source code, including app code, scripts and infrastructure definitions.
Its job is the keep code safe, version it, and provide facilities around handling multiple people committing code to the same items.
Next, Jenkins is going to be our build system. A build system watches the repository and triggers whatever build is required when it changes. In our case, build definition is available within the Dockerfile, so Jenkins job is to watch for changes and build a new image and then push it to the artifact repository: Dokcr Hub.
Ideally, there needs to be some sort of testing after every step but given the short period, we excluded this essential principle of DevOps.

3.2. Working with Jenkins

Jenkins is a self-contained, open-source automation server which can be used to automate all sorts of tasks related to building, testing, and delivering or deploying software.

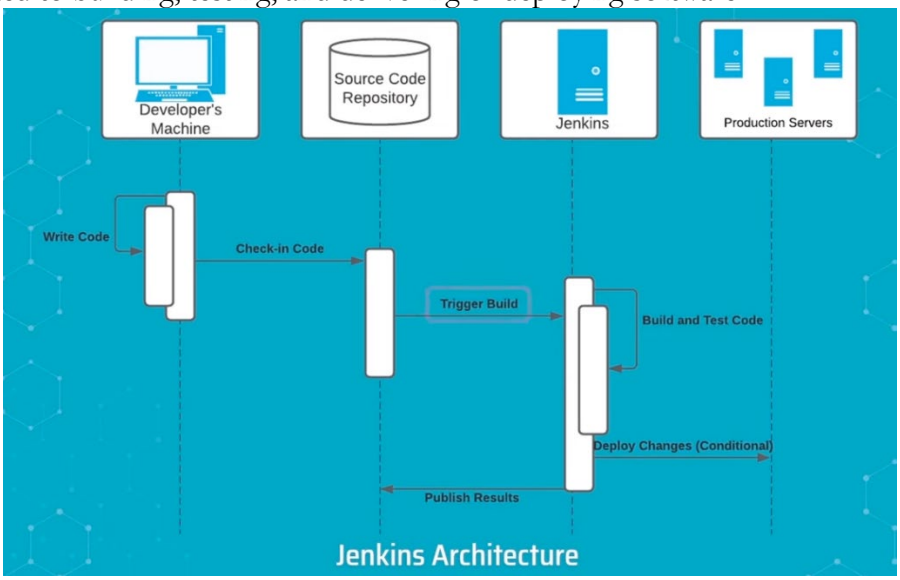


Figure 12: Jenkins architecture [4]

Jenkins Pipeline is a suite of plugins which supports implementing and integrating continuous delivery pipelines into Jenkins.
The definition of a Jenkins Pipeline is typically written into a text file called a Jenkinsfile.
Our Jenkinsfile:

```
pipeline {
  environment {
    imagename = "alaeddinehamroun/odoo"
    registryCredential = 'dockerhub'
    dockerImage = ''
  }
  agent any
  stages {
    stage('Cloning Git') {
      steps {
        git([url:
'https://alaeddinehamroun@bitbucket.org/alaeddinehamroun/odoo_docker.git',
branch: 'master', credentialsId: 'bitbucket'])

      }
    }
    stage('Building image') {
      steps{
        script {
          dockerImage = docker.build imagename
        }
      }
    }
    stage('Deploy Image') {
      steps{
        script {
          docker.withRegistry( '', registryCredential ) {
            dockerImage.push("1.$BUILD_NUMBER")
             dockerImage.push('latest')
          }
        }
      }
    }
    stage('Remove Unused docker image') {
      steps{
        sh "docker rmi $imagename:1.$BUILD_NUMBER"
         sh "docker rmi $imagename:latest"

      }
    }
  }
}
```

Of course, for this to work, we need to setup a webhook from bitbucket and define docker hub credentials into Jenkins.

4. Configuration management with Ansible:

4.1. What is Ansible?

Configuration Management is the process of maintaining systems, such as computer hardware and software, in a desired state. Configuration Management (CM) is also a method of ensuring that systems perform in a manner consistent with expectations over time.

Ansible is an IT automation tool. It can configure systems, deploy software, and orchestrate more advanced IT tasks such as continuous deployments or zero downtime rolling updates. For the most part, the purpose of Ansible is to provision servers.
Ansible manages machines in an agent-less manner. It uses OpenSSH — the open-source connectivity tool for remote login with the SSH (Secure Shell) protocol.

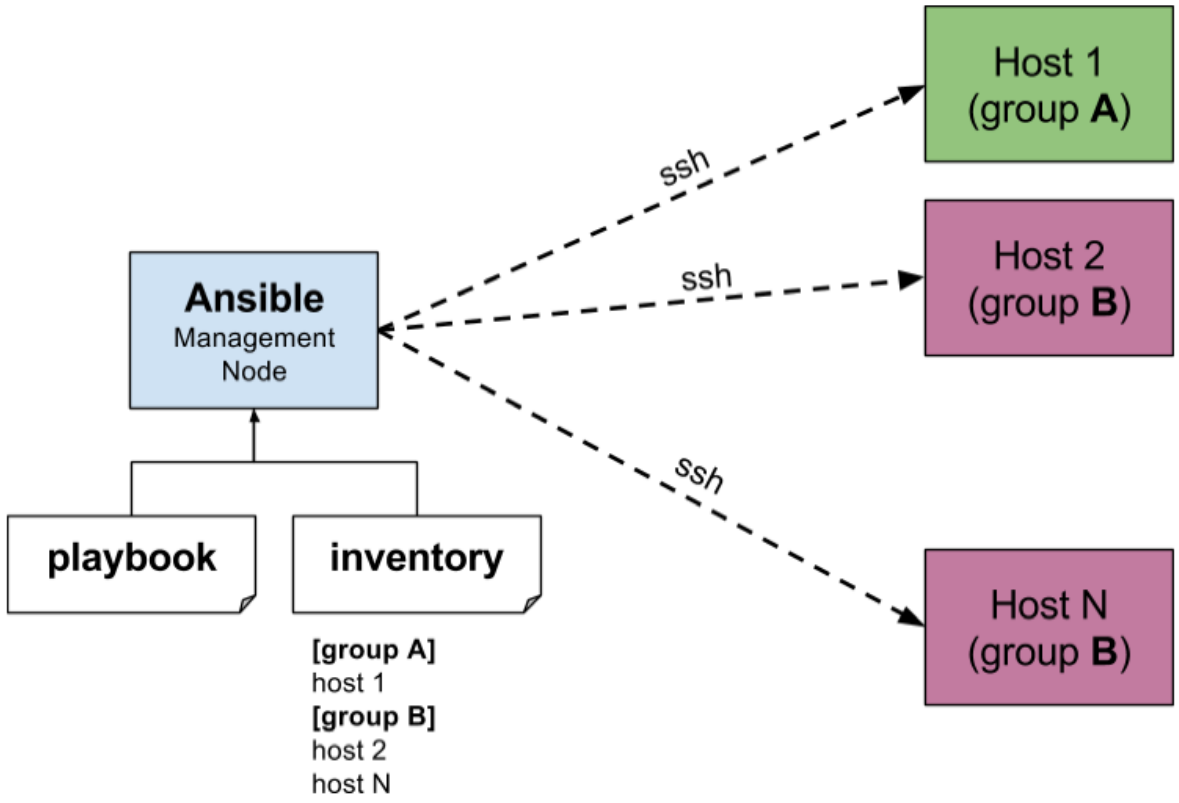


Figure 13: Ansible architecture [5]

There is no one way to setup Ansible.

```
ssh-keygen -t ed25519 -C 'ansible'
```

4.2. Working with Ansible

The first step is to make sure OpenSSH is installed on our workstation and servers.
The following command installs OpenSSH:

```
sudo apt install openssh-server
```

To see the folder where ssh keys will be stored:

```
ls -la ~/.ssh
```

To generate a ssh key specific to ansible:

```
ssh-keygen -t ed25519 -C 'ansible'
```

[-C]: Comment
[-t]: type
The ed25519 format is a more secure key than the default.

To send the public ssh key to server:

```
ssh-copy-id -i ~/.ssh/ansible.pub server_ip_address
```

Ansible.cfg file:
Certain settings in Ansible are adjustable via a configuration file (ansible.cfg).

```
[defaults]
inventory = inventory
private_key_file = ~/.ssh/ansible
```

Playbooks:
An Ansible playbook is a blueprint of automation tasks. They are executed on a set, group, classification of hosts, which together make up an Ansible inventory.
Inventory file contains the ip addresses of the hosts.
I've used Ansible to do the following:

- Docker-config:

Install_docker.yml playbook:

```yaml
---
- hosts: all
  become: true

  tasks:
    - name: Update repository index
      apt:
        update_cache: yes
    - name: Install apt-transport-https
      apt:
        name:
          - apt-transport-https
          - ca-certificates
          - lsb-release
          - gnupg
        state: latest
        update_cache: true


    - name: Add signing key
      apt_key:
        url: "https://download.docker.com/linux/ubuntu/gpg"
        state: present

    - name: Add repository into sources list
      apt_repository:
        repo: "deb https://download.docker.com/linux/ubuntu focal stable"
        state: present

    - name: Install Docker
      apt:
        name:
          - docker
          - docker.io
          - docker-compose
          - docker-registry
        state: latest
        update_cache: true

    - name: Ensure group docker exists
      group:
        name: docker
        state: present
    - name: Ensure user docker exists
      user:
        name: docker
        group: docker

    # reset_connection doesn't support conditionals.
    - name: Reset connection so docker group is picked up.
      meta: reset_connection
```

Run_docker_compose.yml playbook:

```
---

- hosts: all
  become: true
  tasks:
    - name: copy docker-compose.prod.yml file to prod server
      copy:
        src: ../../docker-compose.prod.yml
        dest: /usr/local/bin
        owner: root
        group: root
        mode: 0644
    - name: docker compose up
      community.docker.docker_compose:
        project_src: /usr/local/bin
        files:
          - docker-compose.prod.yml
```

- Nginx config:

```
---
- hosts: all
  become: true
  tasks:
    - name: ensure nginx is at the latest version
      apt:
        name: nginx
        state: latest
    - name: start nginx
      service:
        name: nginx
        state: started
        enabled: yes
    - name: remove default nginx conf file
      file:
        path: /etc/nginx/sites-enabled/default
        state: absent
    - name: copy new nginx conf file to server
      copy:
        src: azure-test.koreacentral.cloudapp.azure.com
        dest: /etc/nginx/sites-available/
        owner: root
        group: root
        mode: 0644
    - name: Create a symbolic link
      file:
        src: /etc/nginx/sites-available/azure-
test.koreacentral.cloudapp.azure.com
        dest: /etc/nginx/sites-enabled/azure-
test.koreacentral.cloudapp.azure.com
        owner: root
        group: root
        state: link

    - name: Restart nginx service
      service:
        name: nginx
        state: restarted
```

- Ssl config:

```
---
- hosts: all
  become: true
  tasks:
  - name: udpate repository index
    apt:
      update_cache: yes

  - name: install python package
    apt:
      name: python3
      update_cache: yes
      state: latest
  - name: install python3-certbot-nginx
    apt:
      name: python3-certbot-nginx
      state: latest
      update_cache: yes

  - name: Generate new certification
    shell: "certbot certonly --standalone --noninteractive --agree-tos --
email alaeddinehamroun@gmail.com -d azure-
test.koreacentral.cloudapp.azure.com"

  - name: Restarting nginx
    service:
      name: nginx
      state: restarted
```

Azure-test.koreancentral.cloudapp.azure.com is the DNS given by Azure for our server.
Azure-test.koreancentral.cloudapp.azure.com:

```
# Odoo Upstreams
upstream odooserver {
server 127.0.0.1:8069;
}

# http to https redirection
server {
    listen 80;
    server_name azure-test.koreacentral.cloudapp.azure.com;
    return 301 https://azure-
test.koreacentral.cloudapp.azure.com$request_uri;
}

server {
    listen 443 ssl;
    server_name azure-test.koreacentral.cloudapp.azure.com;
    access_log /var/log/nginx/odoo_access.log;
    error_log /var/log/nginx/odoo_error.log;

   # SSL
    ssl_certificate /etc/letsencrypt/live/azure-
test.koreacentral.cloudapp.azure.com/fullchain.pem;
    ssl_certificate_key /etc/letsencrypt/live/azure-
test.koreacentral.cloudapp.azure.com/privkey.pem;
    ssl_trusted_certificate /etc/letsencrypt/live/azure-
test.koreacentral.cloudapp.azure.com/chain.pem;


    # Proxy settings
    proxy_read_timeout 720s;
    proxy_connect_timeout 720s;
    proxy_send_timeout 720s;
    proxy_set_header X-Forwarded-Host $host;
    proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
    proxy_set_header X-Forwarded-Proto $scheme;
    proxy_set_header X-Real-IP $remote_addr;

    # Request for root domain
    location / {
       proxy_redirect off;
       proxy_pass http://odooserver;
    }

    # Cache static files
    location ~* /web/static/ {
        proxy_cache_valid 200 90m;
        proxy_buffering on;
        expires 864000;
        proxy_pass http://odooserver;
    }

    # Gzip Compression
    gzip_types text/css text/less text/plain text/xml application/xml
application/json application/javascript;
    gzip on;
}
```

5. Infrastructure monitoring with Zabbix:

5.1. What is Zabbix?

Monitoring refers to the practice of making the performance and status of infrastructure visible. Zabbix is an enterprise-class open-source distributed monitoring solution.

Zabbix is a software that monitors numerous parameters of a network and the health and integrity of servers, virtual machines, applications, services, databases, websites, the cloud and more. Zabbix uses a flexible notification mechanism that allows users to configure e-mail-based alerts for virtually any event. This allows a fast reaction to server problems. Zabbix offers excellent reporting and data visualization features based on the stored data. This makes Zabbix ideal for capacity planning.

Zabbix consists of several major software components.

- Server:

Zabbix server is the central component to which agents report availability and integrity information and statistics. The server is the central repository in which all configuration, statistical and operational data are stored.

- Database storage:

All configuration information as well as the data gathered by Zabbix is stored in a database.

- Web interface:

For an easy access to Zabbix from anywhere and from any platform, the web-based interface is provided. The interface is part of Zabbix server, and usually (but not necessarily) runs on the same physical machine as the one running the server.

- Proxy:

Zabbix proxy can collect performance and availability data on behalf of Zabbix server. A proxy is an optional part of Zabbix deployment. However, it may be very beneficial to distribute the load of a single Zabbix server.

- Agent:

Zabbix agents are deployed on monitoring targets to actively monitor local resources and applications and report the gathered data to Zabbix server.

- Data flow:

In addition, it is important to take a step back and have a look at the overall data flow within Zabbix. In order to create an item that gathers data you must first create a host. Moving to the other end of the Zabbix spectrum you must first have an item to create a trigger. You must have a trigger to create an action. Thus, if you want to receive an alert that your CPU load is too high on *Server X* you must first create a host entry for *Server X* followed by an item for monitoring its CPU, then a trigger which activates if the CPU is too high, followed by an action which sends you an email.

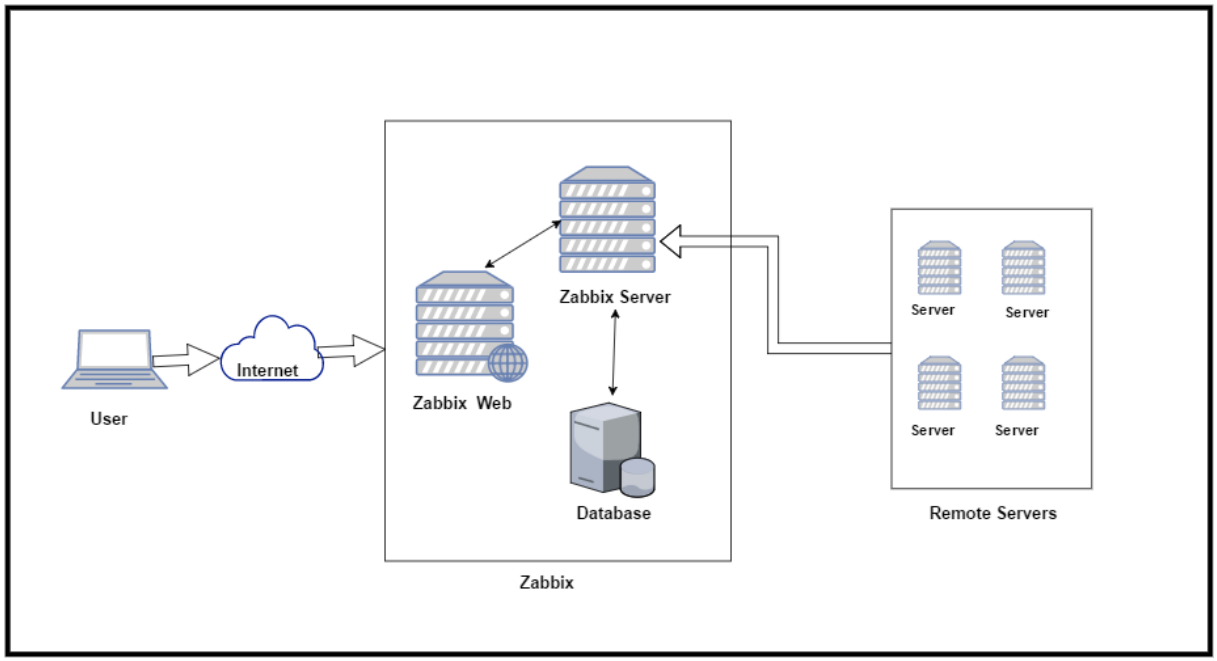


Figure 14: Zabbix architecture [6]

5.2. Setting up Zabbix with Ansible

Install-zabbix-agent.yaml playbook:

```
---
- hosts: all
  gather_facts: yes
  tasks:
      - name: Download zabbix repo package
        get_url:
            url:
"https://repo.zabbix.com/zabbix/6.0/ubuntu/pool/main/z/zabbix-
release/zabbix-release_6.0-1%2Bubuntu20.04_all.deb"
            dest: /tmp/zabbix.deb

      - name: Install zabbix repo
        become: yes
        apt:
            deb: /tmp/zabbix.deb
            state: present

      - name: Install zabbix agent
        become: yes
        apt:
            name: zabbix-agent
            state: present
            update_cache: yes

      - name: Stop service zabbix-agent
        become: yes
        service:
            name: zabbix-agent
            state: stopped

      - name: Remove zabbix config file
        become: yes
        file:
            path: /etc/zabbix/zabbix_agentd.conf
            state: absent

      - name: Create new zabbix config file from template
        become: yes
        template:
            src: "templates/zabbix_agentd.conf.j2"
            dest: "/etc/zabbix/zabbix_agentd.conf"

      - name: Start service zabbix-agent
        become: yes
        service:
            name: zabbix-agent
            state: started
```

V. Conclusion

In a nutshell, this internship has been an excellent and rewarding experience. I can conclude that there have been a lot I've learnt from my work at SOFTIFI. It allowed me to discover how to behave within a company and I acquired new skills concerning putting applications to production such as containerizing an application using Docker, managing servers with Ansible and monitoring them with Zabbix. Needless to say, the technical aspects of the work I've done are not flawless and could be improved provided enough time. As someone with no prior experience with DevOps whatsoever I believe my time spent in research and discovering it was well worth it. Two main thing that I've learned the importance of, are time-management and self-motivation.
I hope to be able to progress well in my professional life.

List of references

[1] Retrieved from: https://www.odoo.com/documentation/15.0/

[2] Retrieved from: https://www.docker.com/resources/what-container/

[3] Retrieved from: https://www.atlassian.com/continuous-delivery/principles/continuous-integration-vs-delivery-vs-deployment

[4] Retrieved from: https://www.youtube.com/watch?v=nCKxl7Q_20I

[5] Retrieved from: https://www.devopsschool.com/blog/understanding-ansible-architecture-using-diagram/

[6] Retrieved from: https://blog.cloudthat.com/zabbix/