

Cloud Engineering Principles

by Edward Patch

Submission date: 26-Apr-2023 05:40PM (UTC+0100)

Submission ID: 203555413

File name: 11299_Edward_Patch_Cloud_Engineering_Principles_488474_29921591.pdf (127.82K)

Word count: 3863

Character count: 22458

Abstract—With the growth of IoT, services and features, cloud computing is vital in today's age. Cloud platforms offer solutions for sharing physical resources with multiple organisations with a few clicks and costs. This research paper focuses on Cloud services' versatility, cost efficiency and scalability when designing, testing and deploying services. Docker and AWS are heavily looked at to understand the benefits and pushbacks of Cloud Computing and the techniques the solutions and techniques offer. Docker containers are used on the Cloud, often providing cost-effective solutions. A Docker container experiment uses Dockerhub, Docker with a server, service and MongoDB to illustrate networking within containers.

Index Terms—Cloud Engineering, Docker, Containers and Virtualisation, Web Services.

I. INTRODUCTION

Technology has changed how we access, develop, deploy and manage our systems, whether it comes to designing a Cloud environment, deploying an application or monitoring any costs. Cloud Engineering is a field evolving quickly, introducing new features and capabilities to give companies control of networking and storage solutions. There are many ways to deploy a system and many platforms. This paper focuses on three leading platforms to understand how containerisation, scalability and modularity are handled.

When establishing the principles, versatility and structures of Cloud computing involving containerisation, including images, volumes and networks, it is essential to understand the roles of containerisation. This ideology includes the accessibility and usability of how containers work and the type of containers that exist. The report will continue onto virtualisation and container managers like Docker.

Expanding the scope of containers within Cloud platforms, the report will continue onto how Docker and DockerHub work, including the perks and cons of Docker with a description of the costs involved.

The research will establish services that enable scalability. These include services such as load balancers and application gateways to control traffic flow to prevent a Virtual Machine (VM) from being overloaded. The research will establish ways of scaling services with automation flows and scripts to control costs and traffic flows.

II. RELATED WORK IN LITERATURE

Cloud providers use virtualisation technology share physical resources with virtual resources. This means that if a for instance, a network card was installed, and it had one ethernet interface connected, then virtualisation shares the connection

to other virtual resources. Due to the abstractness of virtualisation, engineers get this confused with actual physical access, when it is a fake interface module to trick the VM into thinking it is built ontop of a physical machine. The Cloud is built to offer three infrastructure services, Software as a Service (SaaS), Platform-as-a-Service (PaaS) and Infrastructure as a Service (IaaS) through public, private and hybrid cloud models. These infrastructure services provide flexibility to the costs to their consumers. Usually, the less control and abstract the service is, the cheaper and more user experience the service becomes, enabling clients who do not understand Cloud platforms to use with little-to-no effort. Whereas, the PaaS and IaaS allows deeper control and access, in some cases, saving costs compared to SaaS if the product requirements suit the service speciality. [1] [2]

The benefits of virtualisation does a few things for Cloud providers, including the ability to deploy services without the need to deploy physical data-centers and allowing products to deploy much faster. Cloud providers tend to find that virtualisation creates a easier and more effective way to perform maintenance operations. Due to the nature of how Cloud providers setup the infrastructure, they can close certain hardware down and it will not affect service performance or availability, meaning that systems will not collapse, allowing a smooth transition when upgrading hardware or other methods. [3] [2]

Claus Pahl [4] of the Irish Centre for Cloud Computing and Commerce highlights the usefulness of containerisation and how the PaaS clouds rely heavily on virtualisation techniques to provide dedicated services to consumers.

SaaS services is Cloud software already provided and hosted. These require limited configuration and some of example of everyday SaaS services for instance are Office 365, Salesforce, and Google Workspace, providing tools for businesses without extra effort. These services tend to provide tools to increase team production and efficiency without software installations. One issue with SaaS services is that it can sometimes be daunting when selecting necessary SaaS packages that are cost efficient. For example, Salesforce's Customer Relationship Management (CRM) is unaffordable for startups and small businesses, however, with the rarity of other competitors, the Salesforce tends to win over businesses with hope of driving the leads and clients and project management efficiency. AWS, a cloud service provider, offers a SaaS-like element called the AWS Management Console, which allows users to manage their PaaS and IaaS services through

a user-friendly web interface without using the Command-line Interface (CLI). [5] [6]

PaaS services surround building methods, deploying, and managing applications without constructing the initial infrastructure. AWS offers a variety of PaaS services such as Elastic Beanstalk, which enables rapid deployment, management, and scaling of applications; Lambda, a service that runs code at runtime without any additional provisioning, focusing on scaling, monitoring, and maintaining the application; and App Runner, a solution that streamlines the process of building, deploying, and scaling containerized applications. [7]

IaaS services manages the virtual resources available on the Cloud for the consumers needs. These help meet requirements for cost saving, faster deployments, increased reliability and redundancy, and scalability and flexibility. Load Balancing and Application Gateways tools help with the these topics and are classed as IaaS as they manage resources within the Cloud providing a optimised layer for the VMs. IaaS contains services to manage policy control and other requirements, for example, scaling a UK business to serve EU consumers, whilst obeying any security and data laws. [8]

Babak Bashari Rad [9] suggests that Docker containers bring speed, portability, scalability, rapid delivery and density. The benefits of Docker containers is the speed of building and deploying a Docker container. It mentions that during the conversation of Chavis and Architect, 2015 [9] that development, testing and deployment is much faster with the use of Docker containers. The portability of Docker containers is extremely possible, due to the lightweight of containers and the Docker scripts to build and run the application. It is faster to deploy a Docker container than to deploy a VM. Docker has good scalability support and is claimed by Babak Bashari Rad [9] that 'it can be deployed in several physical servers, and cloud platforms. It can also be run on every Linux machine.'. Docker containers can provide rapid delivery with the nature of the application.

III. CLOUD COMPUTING - CONTAINERISATION

The Cloud is versatile, enabling users to create and run multiple VMs that share resources. It allows the creation of numerous VMs, all to share resources. Setting up many resources is daunting. Docker provides a solution enabling scripts to configure the required resources within a container. DockerHub contains a repository system, similar to GitHub, for storing images created with Docker. Although, a common issue Cloud services now had was that engineers had to make a VM to load the Docker container and load in the services, making the process daunting. This service alone saved costs as only a few VMs run the benefits and servers on the VM.

Amazon AWS introduces Elastic Container Service (ECS) to create containers and load them in the Docker configuration to combine. This service reduces the need for extensive structure, allowing developers to focus on writing code and packaging applications in Docker containers. AWS also provides a SaaS interface to keep configuration down, allowing developers to develop under a Docker container, give the

package to the Cloud engineer, and deploy without further configuration. This interface reduces staff costs, saves time and delivers a pricing edge compared to PaaS and IaaS. Although, it is vital to remember that because the Cloud provides a SaaS interface to set up, maintain and monitor the containers, the control panel interacts with PaaS and IaaS to assign and create important VMs and obtain relevant resources. When using abstracted control panels, it is essential to remember that the service interacts with the appropriate Cloud services to initiate the correct service. AWS displays a potential cost to the service when setting up and offering other SaaS products to manage expenses.

The project below uses two fundamental parts of Docker: the build script, 'Dockerfile' and the configuration script, 'docker-compose.yaml'. It is possible to use Docker's CLI. Although, the configuration file allows an organised structure to control each service and target them within the Dockerfile. The setup of the project is visualised in Docker and Docker Compose: https://drive.google.com/file/d/1jgccZty5hN7_NcfFghRNBoOk192B2clb/view

The Docker file contains a script to automate the Docker container using BASH. This script establishes an operating system template using Ubuntu with the foundation software. The hand then has two targets, 'nodeenv' and 'pyenv', that install the relevant development and production frameworks on top of the developer-os target. Another two targets are initialised called 'client' and 'server', which interact with docker-compose to create the relevant environments. The Dockerfile contains configurations with the VM and networking of what the scripts should target. However, in this case, the docker-compose composition controls the port forwarding and initialisation of the network.

Dockerfile:

```
FROM ubuntu as developer-os
LABEL name="Edward Patch"
email="crazygaming055@gmail.com"
ENV DEBIAN_FRONTEND noninteractive
RUN apt-get update --fix-missing && \
    apt-get install -y \
    software-properties-common && \
    apt-get install -y \
    --no-install-recommends \
    apt-utils && \
    apt-get install -y curl \
    wget
RUN apt-get install -y sudo
RUN echo "export \
SERVER_IP_ADDRESS=0.0.0.0" >> \
/etc/profile
RUN apt-get clean

FROM developer-os as nodeenv
LABEL author="Edward Patch"
email="crazygaming055@gmail.com"

# Create and change the working
directory
WORKDIR /var/www/node
RUN curl -fssl \
https://deb.nodesource.com/setup_19.x
```

```

| sudo -E bash && \
apt-get install -y nodejs
RUN npm install -g npm && \
  npm install -g npx --force
RUN apt-get clean

COPY Client/package.json ./
RUN npm install

#### THE WEB Client #####
FROM nodeenv AS client
EXPOSE 1500
WORKDIR /var/www/node
ENTRYPOINT ["npm", "run", "dev"]

FROM developer-os as pyenv
WORKDIR /var/www/python
RUN add-apt-repository
  ppa:deadsnakes/ppa
RUN apt-get install -y python3.10
RUN curl -sS
  https://bootstrap.pypa.io/get-pip.py
  | python3.10
RUN python3.10 -m pip --version
RUN python3.10 -m pip install --upgrade
  pip
RUN apt-get install -y 3.10-venv
RUN apt-get update | apt-get upgrade -y
RUN pip3 install virtualenv
RUN python3 -m venv ./venv/

COPY Server/requirements.txt ./
RUN . ./venv/bin/activate | pip3
  install -r ./requirements.txt
RUN export FLASK_APP=app

#### THE WEB Server #####
FROM pyenv AS server
EXPOSE 3000
WORKDIR /var/www/python
RUN . ./venv/bin/activate
ENTRYPOINT [ "gunicorn", "-w", "2",
  "-b", "0.0.0.0:3000", "app:app" ]

```

The docker-compose component does a few things. First it creates three containers and a network, 'client', 'server' and 'mongodb'. The docker-compose stores 'client' and 'server' container images onto Dockerhub under the repository, 'cloud-computing' under the username, '1801492'. The two containers point to the Dockerfile to build the containers. The docker-compose file is in charge of selecting the current filing system, volume, and moving it to the containers volume for the Dockerfile to interact with. The network allows communication between each container, allowing API communication with each other. The client uses port 1500, whereas the server uses port 3000.

The docker-compose configuration initiates MongoDB and MongoDB Express, the database hosted on port '27017' and the client hosted on port '8081'. The database root username is set to 'Docker', the root password is set to 'password', and the database name is set to 'mydb'. The Mongo Express connects to the root, allowing users to log on to create collections

and manage data within the database. The 'clint', 'server', 'mongodb' and 'mongodb-express' targets are essential to show the network of the Docker container working. Please note that the Docker container network is virtual and uses the virtualisation network of the VM. Although, Docker container networks are robust when creating and handling ongoing and outgoing communication.

docker-compose.yml:

```

version: "3.6"
services:
  client:
    build:
      context: .
      Dockerfile: Dockerfile
    target: client
    image:
      1801492/cloudcomputing:nodejs-ai
    restart: always
    ports:
      - "1500:1500"
    networks:
      - dbnetwork
    volumes:
      - ./Client:/var/www/node
      - /var/www/node/node_modules
  server:
    build:
      context: .
      Dockerfile: Dockerfile
    target: server
    image:
      1801492/cloudcomputing:pyflask-ai
    restart: always
    ports:
      - "3000:3000"
    volumes:
      - ./Server:/var/www/python
    networks:
      - dbnetwork
  mongodb:
    image: mongo:latest
    restart: always
    ports:
      - "27017:27017"
    environment:
      MONGO_INITDB_ROOT_USERNAME: "Docker"
      MONGO_INITDB_ROOT_PASSWORD:
        "password"
      MONGO_INITDB_DATABASE: "mydb"
    networks:
      - dbnetwork
  mongo-express:
    image: mongo-express:latest
    restart: always
    ports:
      - "8081:8081"
    environment:
      ME_CONFIG_MONGODB_ENABLE_ADMIN:
        "true"
      ME_CONFIG_MONGODB_SERVER: "mongodb"
      ME_CONFIG_MONGODB_ADMINUSERNAME:
        "Docker"
      ME_CONFIG_MONGODB_ADMINPASSWORD:

```

```

    "password"
    ME_CONFIG_MONGODB_PORT: 27017
    depends_on:
      - mongodb
    networks:
      - dbnetwork

# Creating a Network: This "dbnetwork"
# is used to connect the database and
# Server to the same network for
# communication.
networks:
  dbnetwork:

```

Figure 1 shows a NodeJS express web server serving the web service's HTML page on port 1500. The website serves a welcome page with link to the web server's Web API.

Welcome to the Docker website!

- [Backend - http://localhost:3000](http://localhost:3000)

Welcome!

Thank you for visiting my website.

© 2021 MyWebsite.com

Fig. 1. Docker Containerisation: Client

Figure 2 shows a Python Flask web server serving the web server's API on port 3000. The website provides a GET API request to tell the user a welcome message, with the client's web address.

```

{"message": "Welcome to the Flask REST API", "site-details":
{"client": "http://localhost:1500", "status": 200}, "status": 201}

```

Fig. 2. Docker Containerisation: Server

IV. CLOUD COMPUTING - CONCEPTS

A. Load Balancing

Load balancing, aka. Amazon Elastic Load Balancer (ELB) helps control the traffic flow to different resources. Load balancers can direct regional traffic to the appropriate resources closest to the regions of the incoming traffic. This IaaS architecture has different setups and strategies to suit different Product Requirements and Use Cases.

An approach to achieving performance and user experience is using latency-based routing or geolocation-based routing. These methods focus on analysing the source of incoming traffic and re-routing the traffic to the suitable resources in different regions. In accordance of GDPR laws in UK and EU, load balancing is effective routing different information into the right data centers.



Although, business owners based in the UK worry as API's require delivering the data to the load balancer before it is redirected, then any data that's leaked could breach Data Protection Act 2018 and UK GDPR laws for example. This requires security frameworks to be properly researched and implemented, otherwise if data is leaked outside of the UK, the business may suffer repercussions and the cloud provider is not at any fault, unless proof can demonstrate that the security of API's and infrastructure in place were carefully researched, implemented and overviewed. The security in place will stop data breaches and leakage during the load balancing process.

Load Balancing systems eliminate risk on the Cloud system as they monitor the health of different resources, and carefully direct the flow of traffic to the right resources. This IaaS feature can check out the load of each resource, to evenly distribute the flow of traffic.

B. Scalability

Cloud providers always try to manage costs, scalability and user experience. Cloud services optimise scalability across all the services, including PaaS and IaaS. Cloud services know that their services are designed for experienced Cloud Engineers. However, they keep the market open by providing user-friendly tools to open services. These include Load Balancing, Auto-Scaling and Sharding. These tools enable businesses to develop and deploy products for the chosen target audience.

As mentioned previously, load balancing focuses on traffic flow, directing clients to the appropriate service regarding the region and health checks of the server. Scalability requires Load Balancing as a fundamental component to maintain client security, stability and performance, significantly when the Cloud infrastructure and traffic expands.

Auto-scaling is a cost-effective strategy. It enables VMs to start up in anticipation based on current traffic flows. These systems can contain configurations for deeper analytics to predict clients' futures. Either way, auto-scaling means that the user will not pay for a fixed rate across the year for resources that will never be used. According to 'Evaluating Auto-scaling Strategies for Cloud Computing Environments' by Macro A. S. Netto [10] tests the Autoscaling Demand Index (ADI) strategy to scale up the number of VMs automatically. The strategy takes into account the current demand for resources and available capacity. This method is an excellent selection as the strategy recognises peak times and provides sufficient resources to support auto-scaling cost-effectively.

The ADI strategy can improve the performance of the Cloud environment by automatically scaling the number of VMs to match the demand. The ADI strategy can save costs of the Cloud environment by only provisioning the needed resources. The ADI strategy has increased agility making it easy to scale up and down the number of resources. However, this strategy has three central concerns involving complexity to implement and manage, allocating additional bandwidth in network planning and design and downtime, concerning if VMs do not scale up and down quickly to meet demand.

Sharding breaks an extensive database into smaller, manageable pieces of data. This service enhances the performance of the technological system when selecting the most relevant data. An example of sharding is when an organisation have a collection of available drivers in different regions. Splitting the database into regional shards reduces the time it takes to execute and query for available Uber drivers in the client's area drastically. However, sharding is usually applied on matured systems and is usually costly. These include hardware, software and operational costs. Another critical point is that shards have security and development costs and issues when trying to maintain.

C. Application Gateways

Application Gateways, an implementation from the application level (Level 7) of the OSI Model within the networking layers. Application Gateways process the incoming and outgoing requests to what the client and servers receive. Application Gateway's primary role is a security measure. Although, the Application Gateway can focus on load balancing and traffic management for the web application.

Application Gateways communicate with load balancers to anticipate any redirects or new addresses with the server. Application Gateways can figure out routing configurations to make sure the relative pathways are suitable with the physical pathways.

Application Gateway's security frameworks could include Web Application Firewalls (WAF) to protect applications from web-based threats such as Structured (SQL) Injection, Cross-Site Scripting (XSS) and Distributed Denial-of-Service (DDOS). The Application Gateway can examine the content of HTTP/HTTPS requests and responses. This technique can identify and block malicious traffic before it reaches the application servers. Implementation of a Application Gateway supplies controllability, integrity and elasticity allowing it to suit both scalability and security. [11]

Some of the key values of a Application Gateway involves SSL/TLS termination, session persistence, caching and integration with other Cloud services.

SSL/TLS Termination can offload SSL/TLS encryption and decryption tasks from the backend servers. The method reduces any processing overhead on those servers and improves the overall performance. The gateway also includes a method to centralise management of SSL/TLS certificates. Application Gateways have potential to use caching capabilities, providing a method to store and serve frequently accessed content directly, reducing the load on backend servers and improving response times. Application Gateways allow integration with other Cloud services, including monitoring and logging tools, Content Delivery Networks (CDNs) and Auto-Scaling groups.

V. CONCLUSION

Clouds supply SaaS, PaaS and IaaS services to give dynamic cost and control over the Cloud. Platforms are competitive within pricing standards and constantly update and release new services. Clouds use virtualisation to share physical hardware,

which is more cost-friendly than building a physical data centre to create a Cloud application.

It is recommended to stay vigilant around Cloud services as they offer a SaaS configuration panel and a CLI to manage the virtual resources. Cost saving requires knowledge of shutting down VMs or other resources and properly removing them when the service is no longer required.

Scaling saves costs and is automated using Load Balancing, Docker, Application Gateways, and other Cloud services. However, implementing these Cloud services may cost time and funds to run efficiently. It is essential to plan what services the application successfully requires before setting up the services. By thoroughly understanding the application's needs and selecting the appropriate cloud services, businesses can achieve optimal performance, cost savings, and adaptability in an ever-changing technological landscape.

VI. TERMINOLOGY

List of terminologies used in this document:-

- VM - Virtual Machine.
- SaaS - Software-as-a-Service.
- PaaS - Platform-as-a-Service.
- IaaS - Infrastructure-as-a-Service.
- CLI - Command-line Interface.
- ECS - Elastic Container Service.
- ELB - Elastic Load Balancer.
- ADI - Autoscaling Demand Index.
- WAF - Web Application Firewall.
- XSS - Cross-Site Scripting.
- DDOS - Distributed Denial-of-Service.
- CDN - Content Delivery Network.

VII. REFERENCE LIST

- [1] S. Rajan and A. Jairath, "Cloud Computing: The Fifth Generation of Computing," in *International Conference on Communication Systems and Network Technologies*. IEEE, 2011.
- [2] S. Hardikar, P. Ahirwar, and S. Rajan, "Containerization: Cloud Computing based Inspiration Technology for Adoption through Docker and Kubernetes," in *2021 Second International Conference on Electronics and Sustainable Communication Systems (ICESC)*, 2021, pp. 1996–2003.
- [3] S. R. Meadusani, "Virtualization Using Docker Containers: For Reproducible Environments and Containerized Applications," Ph.D. dissertation, Department of Information Systems, St. Cloud State University, May 2018.
- [4] C. Pahl, "Containerization and the PaaS Cloud," *IEEE Cloud Computing*, vol. 2, no. 3, pp. 24–31, 2015.
- [5] W.-T. Tsai, X. Bai, and Y. Huang, "Software-as-a-service (SaaS): perspectives and challenges," *Science China*, vol. 57, no. 5, pp. 1–15, 2014, publisher: Springer.
- [6] V. Kumar, S. V. Jasola, and P. Vidhyalakshmi, "SaaS as a business development tool," in *Conference Proceedings of International Conference on Business Management (ICOBM, 2013)*, University of Management and Technology, Lahore, Pakistan, 2013.
- [7] A. J. Ferrer, D. G. Pérez, and R. S. González, "Multi-cloud Platform-as-a-service Model, Functionalities and Approaches," *2nd International Conference on Cloud Forward: From*

- Distributed to Complete Computing*, vol. 97, pp. 63–72, Jan. 2016. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S187705091632097X>
- [8] F. Wulf, T. Lindner, S. Strahringer, and M. Westner, "IaaS, PaaS, or SaaS? The Why of Cloud Computing Delivery Model Selection: Vignettes on the Post-Adoption of Cloud Computing," in *Proceedings of the 54th Hawaii International Conference on System Sciences, 2021*, 2021, pp. 6285–6294.
 - [9] B. B. Rad, H. J. Bhatti, and M. Ahmadi, "An introduction to docker and analysis of its performance," *International Journal of Computer Science and Network Security (IJCSNS)*, vol. 17, no. 3, p. 228, 2017, publisher: International Journal of Computer Science and Network Security.
 - [10] M. A. Netto, C. Cardonha, R. L. Cunha, and M. D. Assuncao, "Evaluating Auto-scaling Strategies for Cloud Computing Environments," in *2014 IEEE 22nd International Symposium on Modelling, Analysis & Simulation of Computer and Telecommunication Systems*, Sep. 2014, pp. 187–196, iSSN: 2375-0227.
 - [11] Y. Imai, Y. Sugieue, Y. Hori, Y. Iwamoto, and S. Masuda, "An Enhanced Application Gateway for some Web services to Personal Mobile Systems," in *International Conference on Computational Intelligence for Modelling, Control and Automation and International Conference on Intelligent Agents, Web Technologies and Internet Commerce (CIMCA-IAWTIC'06)*, vol. 2, Nov. 2005, pp. 1055–1060.

Cloud Engineering Principles

GRADEMARK REPORT

FINAL GRADE

61 / 100

GENERAL COMMENTS

Instructor

Docker Overview + Benefits (20): 13

Docker Examples (30): 26

Video (5): 5

Cloud Comp tech (25): 17

Cloud Comp functions (20): 0

Feedback:

=====

I wanted a bit more discussion of the benefits of docker with good examples.

The discussion of load balancing is a little misleading and main reason for the app gateway is not really explored i.e. hiding the internal infrastructure.

Could not find a discussion of the factors driving cloud and serverless functions.

PAGE 1

PAGE 2

PAGE 3

PAGE 4



Comment 1

How is this "load" balancing?

PAGE 5

PAGE 6
