Development of a Cloud Platform for Big Geospatial Data Analytics

MASTERS OF TECHNOLOGY
in
Geo-Informatics and Natural Resource Engineering,
Centre of Studies in Resource engineering

Presented By:
Farheen Bano
193310022

Guided By:
Prof. Surya Durbha

Aim

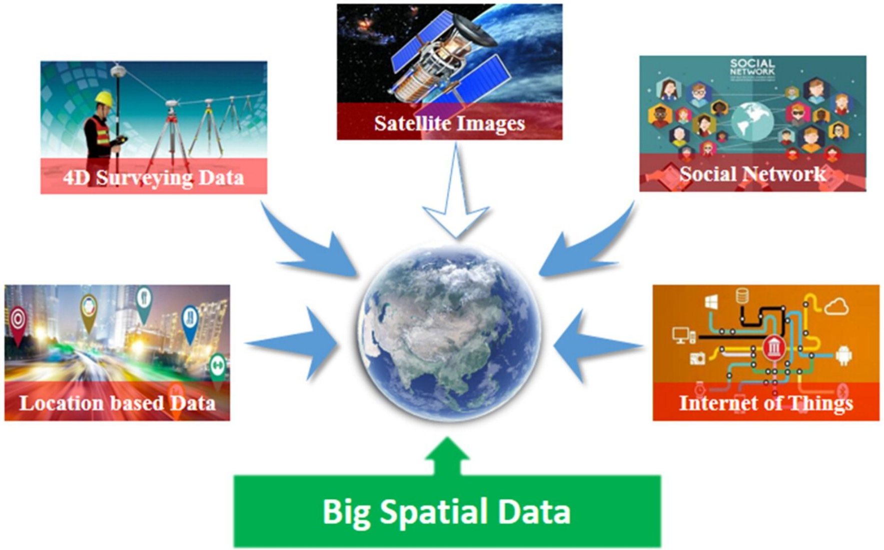
- Advanced **Earth observation** technologies nowadays produce a variety of huge datasets in the form of satellite images. To derive timely information from these datasets, remote sensing scientists need to be equipped with better and powerful processing, computing, and storage platforms.
- **Cloud computing** platforms are a good option since they provide the required computing power with the lowest cost on a pay-as-use basis.
- **Containers** are a solution to the problem of how to get the software to run reliably when moved from one computing environment to another. This could be from a developer's laptop to a test environment, from a staging environment into production, and perhaps from a physical machine in a data center to a virtual machine in a private or public cloud.
- The load balancing, scaling, and orchestrating of the containers can be managed by **Kubernetes**

Big Geospatial Data Sources

- Big Geospatial Data are collected everyday in several forms from these data sources includes to **volume** and **variety**

- Big Geospatial Data is characterised **multi-sourced** as **nonstationary massive, heterogeneous, multi-scale, multi-temporal complicated, and unstructured.**

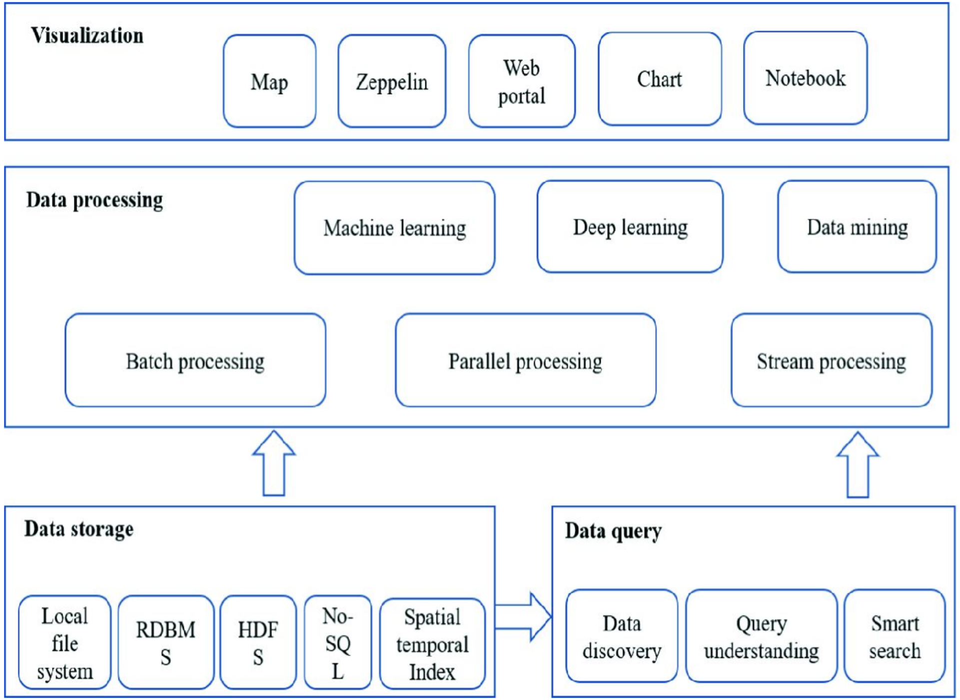
- Big geospatial data can be categorized into three forms: raster data, vector data, and graph data.



The classification diagram of big geospatial data

https://www.tandfonline.com/doi/full/10.1080/20964471.2018.1432115

Architecture of Big Geospatial Data Analytics

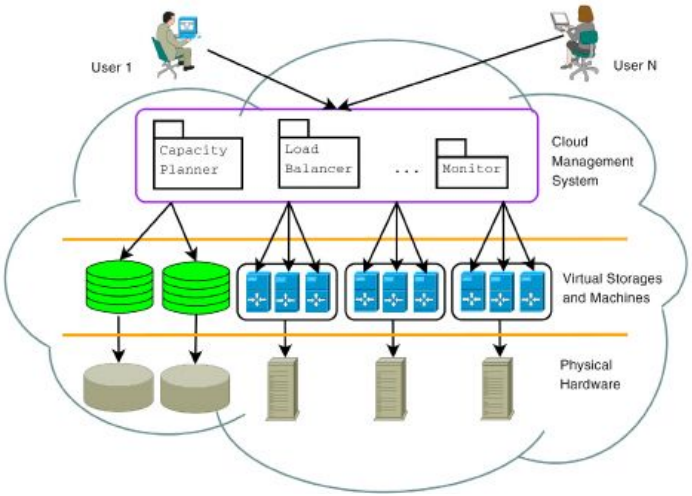


Architecture of big geospatial data analysis

https://link.springer.com/chapter/10.1007/978-981-32-9915-3_9

- **Visualizing** the big spatial data-set, OGC Web Map Service (WMS) has provided an easy solution of displaying maps.
- To examine terabyte and petabyte datasets with reduced time latency, in a real-time fashion, advanced parallel computing algorithms and scalable computing tools are needed in the significant **data processing** framework
- **Spatial database** is an effective means to manage vector data, which is the basis of vector data query, analysis, and application.
- **Spatial Query** operations would be the basis of spatial analysis and GIS application system

Cloud Computing

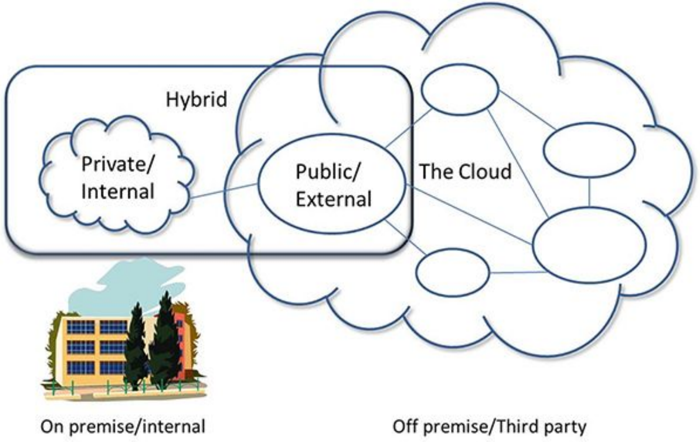


Overview of Cloud Computing

http://gray.biji.us/cloud-architecture/

Cloud computing is the **on-demand** availability of computer system resources, especially data storage and computing power, without direct active management by the user. It has the characteristics of **elasticity**, **pooled resources,** on-demand access, self explanatory and pay-as-you-go characteristics (Mell and Grance 2011) and was termed spatial cloud computing in the context of Digital Earth

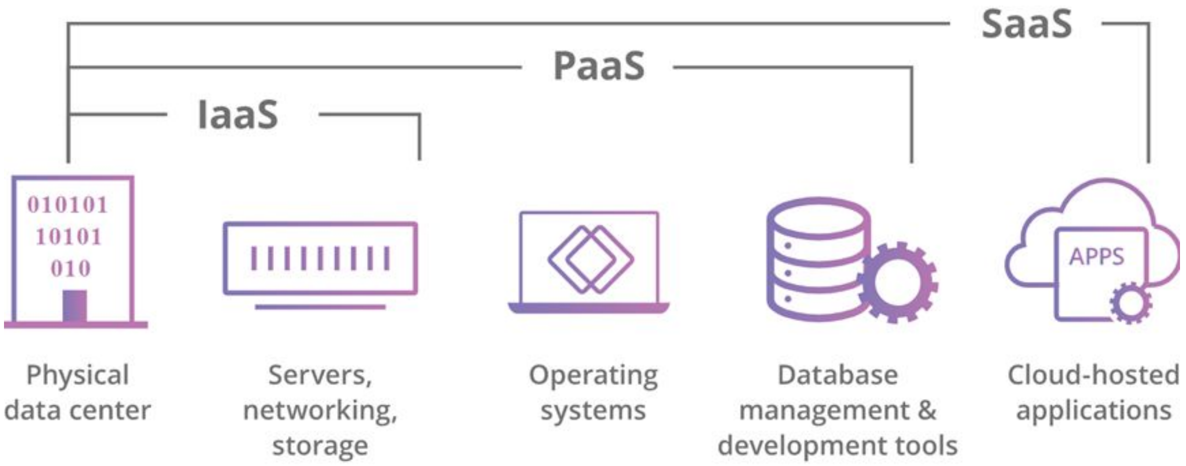
- **Public cloud** is available to the public and might be provided on a pay-per-usage mode.
- **Public cloud** is dedicated for use inside a company.
- **Hybrid cloud** is a composition of a public cloud and a private cloud, offering the benefits of multiple deployment models.



Cloud Types

https://www.industrios.com/news/read/top-5-reasons-hybrid-erp

Cloud Services



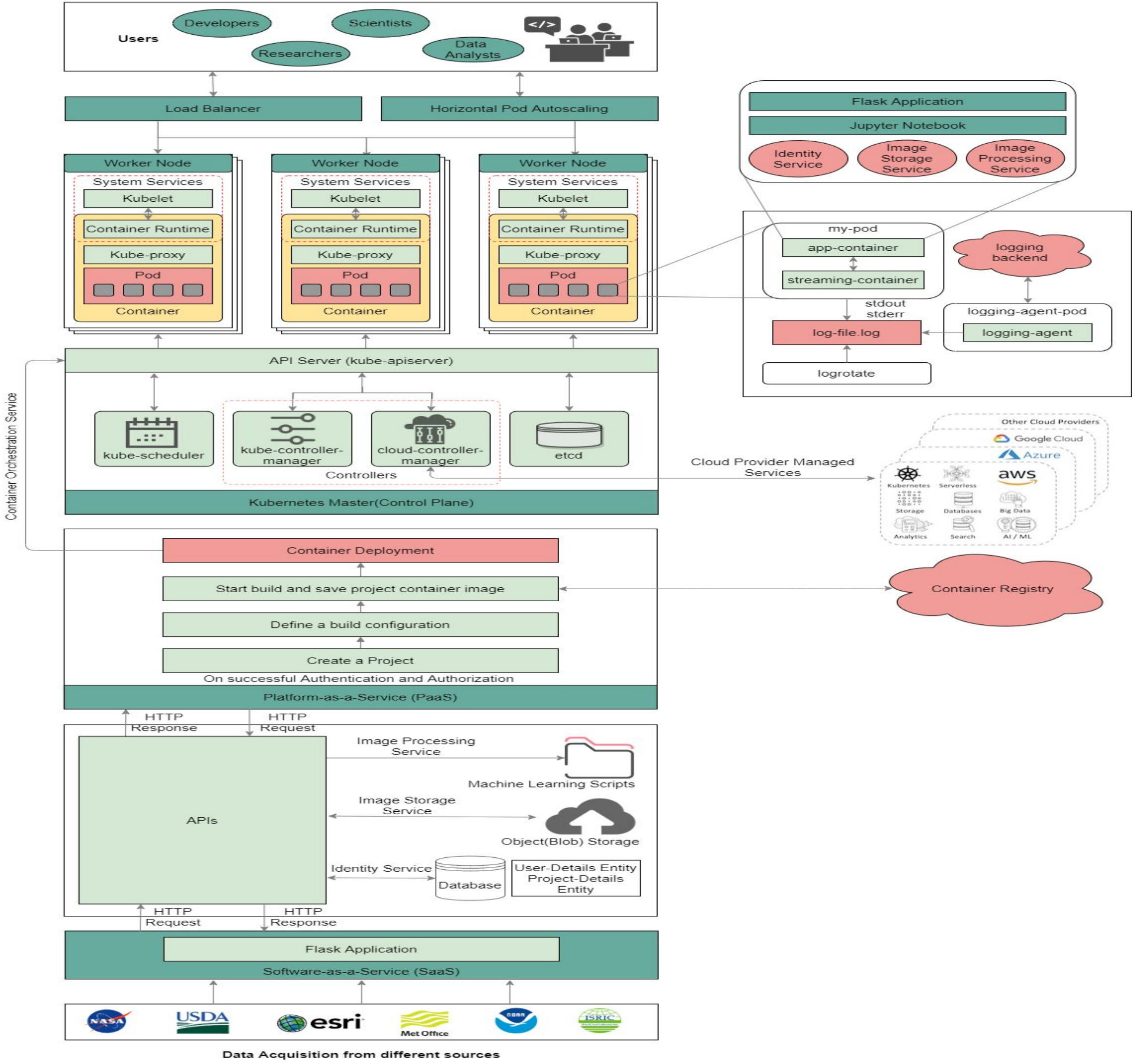
Developing a Cloud based Platform

https://www.cloudflare.com/learning/serverless/glossary/platform-as-a-service-paas/

IaaS: cloud-based services, pay-as-you-go for services such as storage, networking, and virtualization.
PaaS: hardware and software tools available over the internet.
SaaS: software that's available via a third-party over the internet.

Methodology

Depending on the papers reviewed and conclusions inferred from it, an Architecture for the Development of a Cloud Platform for Big Geospatial Data Analytics is proposed. The proposed architecture includes all the components required for the development of a cloud platform. All the drawbacks and benefits of using these components for this specific purpose wes taken into consideration.

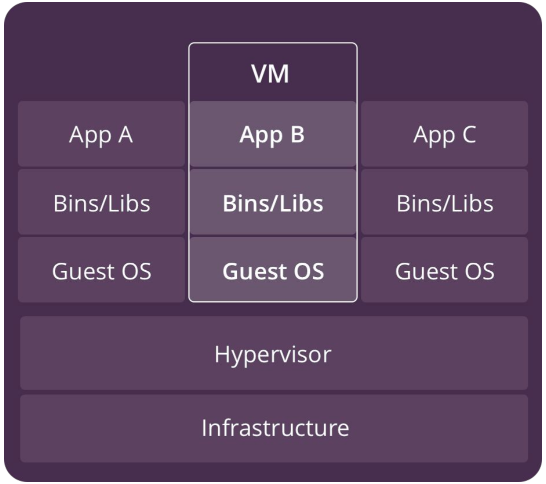


Proposed architecture

for

"Development of a Cloud Platform for Big Geospatial Data Analytics"

Virtualization

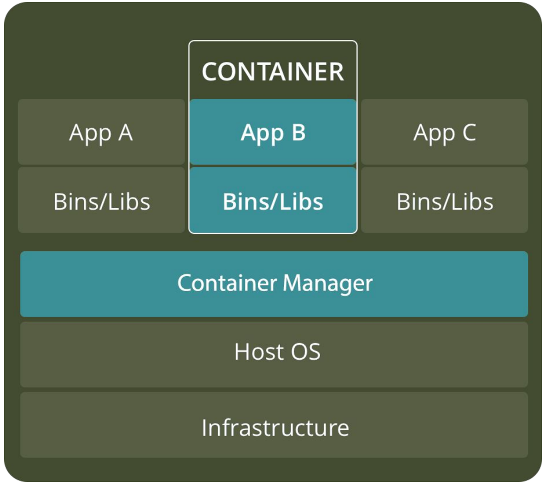


Virtualization and Virtual Machines

https://www.backblaze.com/blog/vm-vs-containers/

- **Virtualization** creates an abstraction layer over computer hardware that allows the hardware elements of a single computer to be divided into multiple virtual computers.

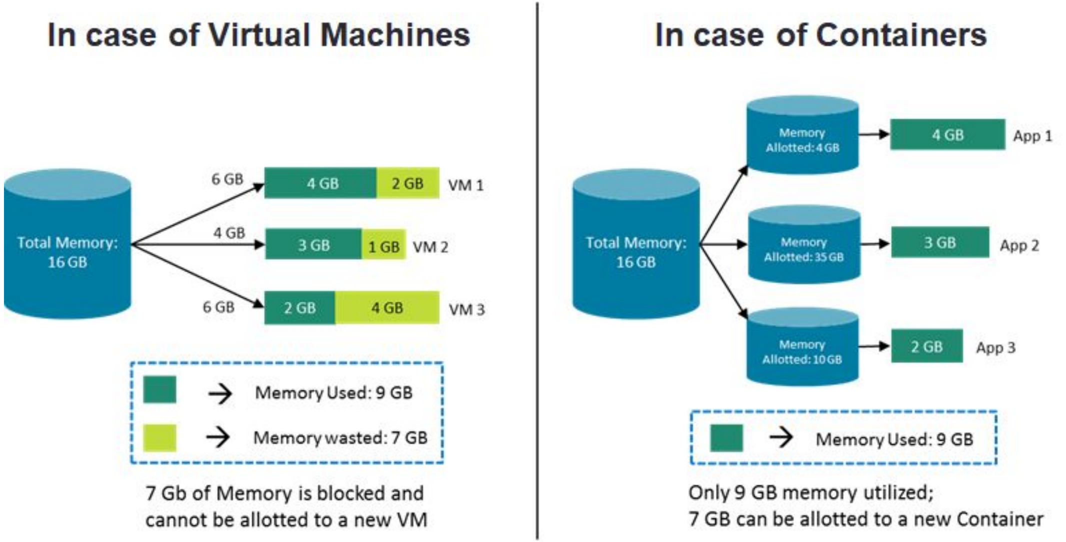
Containerization



Containerization and containers

https://www.backblaze.com/blog/vm-vs-containers/

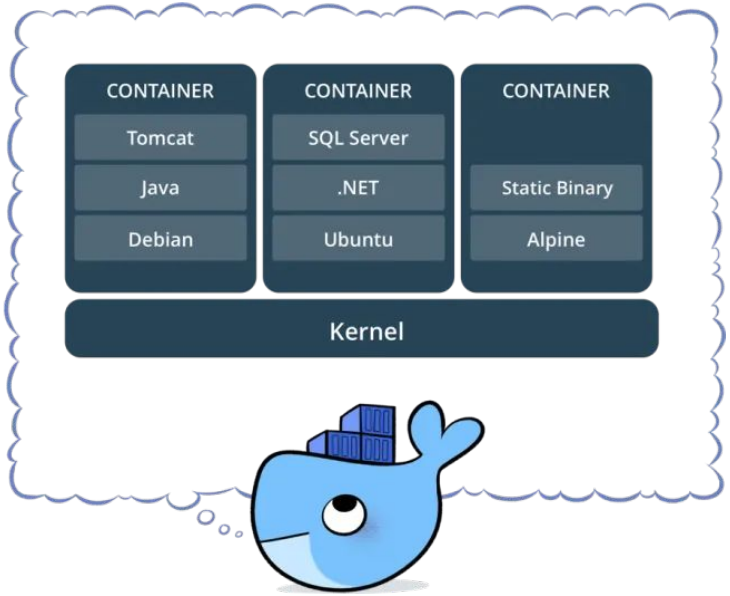
- **Containerization** means encapsulating an application—often a single executable service or microservice—along with its libraries, frameworks and other components.



https://imranhsayed.medium.com/getting-started-with-docker-8eaae6bf2183

Virtual Machine	Container
Heavyweight	Lightweight
Limited performance	Native performance
Each VM runs in its own OS	All containers share the host OS
Hardware-level virtualization	OS virtualization
Startup time in minutes	Startup time in milliseconds
Allocates required memory	Requires less memory space
Fully isolated and hence more secure	Process-level isolation, possibly less secure

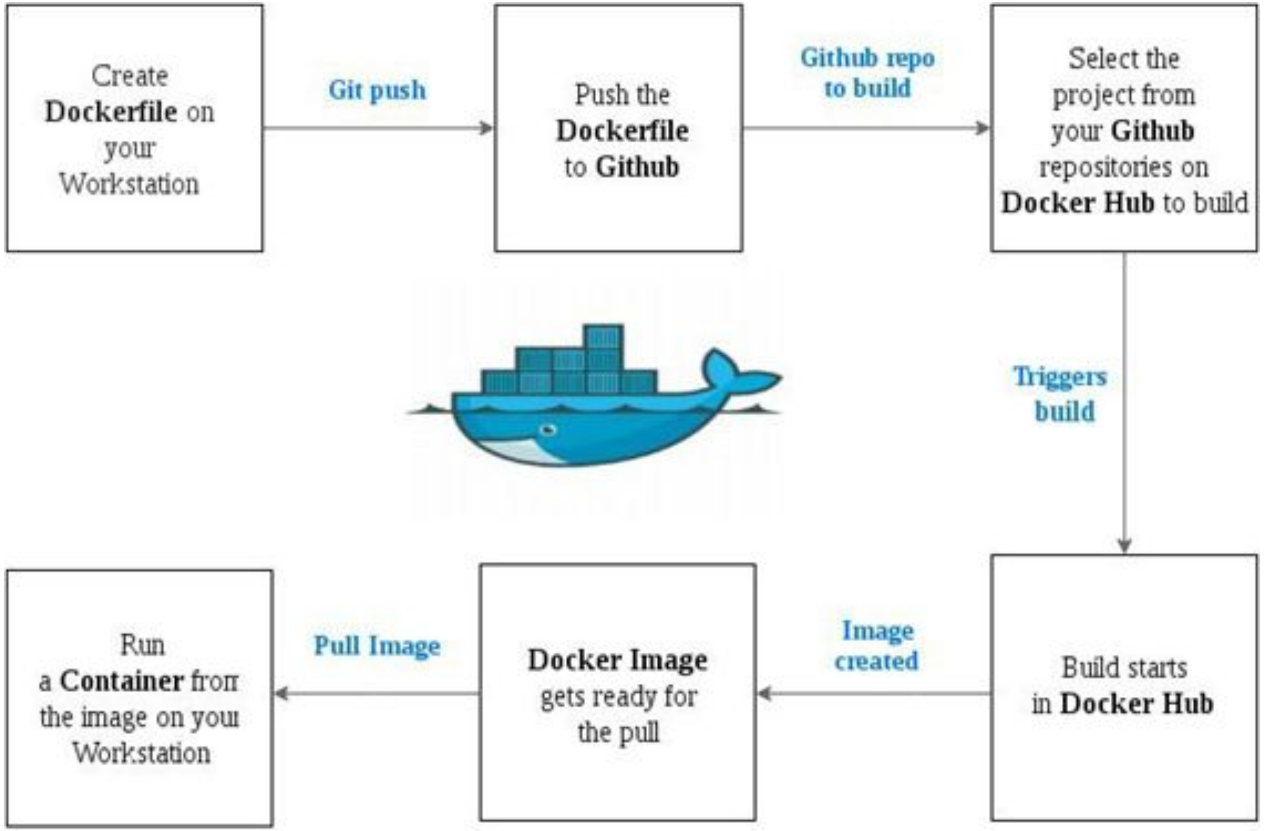
Docker Container



Docker container

https://imranhsayed.medium.com/getting-started-with-docker-8eaae6bf2183

- A Docker container is a **virtualized run-time environment** where users can isolate applications from the underlying system.
- These containers are **compact**, **portable** units in which we can start up an application quickly and easily.



Steps to execute a container

https://medium.com/faun/how-to-build-a-docker-container-from-scratch-docker-basics-a-must-know-395cba82897b

Flow starts with a script of instructions called Dockerfile that defines how to build a specific Docker image. The file automatically executes the outlined commands and creates a **Docker image**.

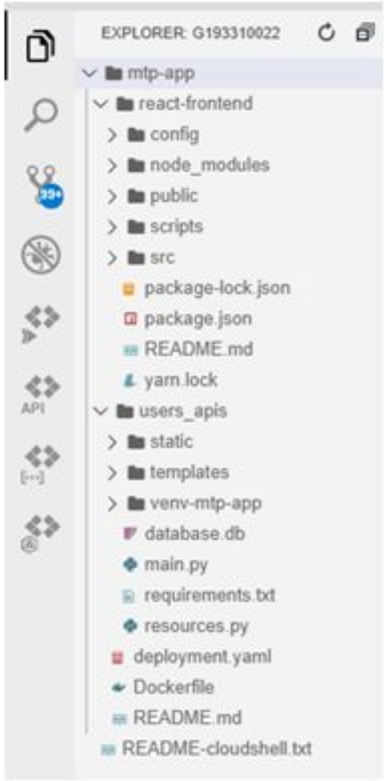
$ docker build -t mtp_geodata_api:latest .

The following Dockerfile used to create this application image:

```
FROM ubuntu:18.04
FROM python:3.7.3
LABEL maintainer="Farheen<farheenbano94@gmail.com>"

#  for Flask backend app
RUN apt-get update -y && apt-get install -y python-pip python-dev
COPY . /app
WORKDIR /app/users_apis
RUN pip install -r requirements.txt

#  for React front-end app
WORKDIR /app/react-frontend
RUN apt-get update && apt-get install -y curl
RUN curl -sL https://deb.nodesource.com/setup_14.x | bash -
RUN apt-get install -y nodejs
RUN npm install
RUN npm run build
WORKDIR /app/users_apis
EXPOSE 5000
CMD [ "python3", "./main.py" ]
```



The folder structure of the application

Command to create docker images:

```
(venv-mtp-app) g193310022@cloudshell:~/mtp-app(mtp-app-318619)$ docker build -t mtp_geodata_api:latest .
Sending build context to Docker daemon  33.37MB
Step 1/16 : FROM ubuntu:18.04
 ---> 7d0d8fa37224
```

Command for viewing docker images:

```
(venv-mtp-app) g193310022@cloudshell:~/mtp-app (mtp-app-318619)$ docker images
REPOSITORY        TAG       IMAGE ID       CREATED         SIZE
mtp_geodata_api   latest    ce44482fc7d1   5 minutes ago   1.69GB
ubuntu            18.04     7d0d8fa37224   2 weeks ago     63.1MB
python            3.7.3     34a518642c76   2 years ago     929MB
```

Command for running a the docker image:

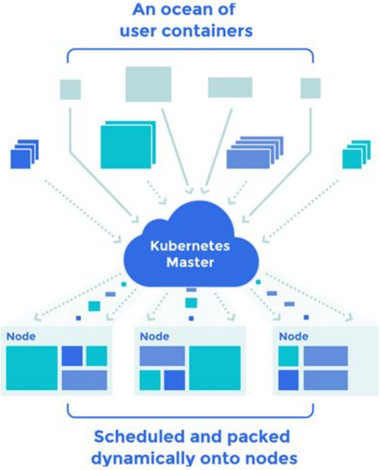
```
(venv-mtp-app) g193310022@cloudshell:~/mtp-app (mtp-app-318619)$ docker run -it mtp_geodata_api:latest
sh
# ls
Dockerfile  deployment.yaml  react-frontend  users_apis
# cd users_apis
# ls
database.db  main.py  requirements.txt  resources.py  static  templates
# cd ..
# cd react-frontend
# ls
config  node_modules  package-lock.json  package.json  public  scripts  src  yarn.lock
# exit
```

Command for checking Container status:

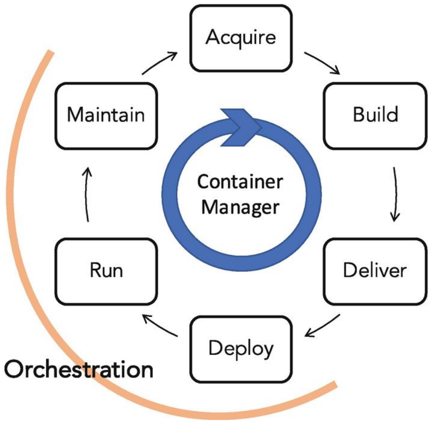
```
(venv-mtp-app) g193310022@cloudshell:~/mtp-app (mtp-app-318619)$ docker ps
CONTAINER ID  IMAGE           COMMAND            CREATED         STATUS       PORTS                   NAMES
66602128b901  mtp_geodata_api "python3./main.py" 10 seconds ago  Up 9 seconds 0.0.0.0:5000->5000/tcp  xenodochial_booth
```

Kubernetes: Container Orchestration

- Kubernetes does the **orchestration** and **management** of containers so that they can be used in a production environment with the proper set of controls.
- Kubernetes does all the work to make sure that this **desired state** is always kept up and running.
- Kubernetes in production includes **monitoring**, **networking**, and **load balancing** so that when demand increases, the application architecture developed with containers scales and meets the demand.
- Kubernetes also provides a way to **update** to container **versions** in a continuous delivery model and rollback to a previous state when necessary.

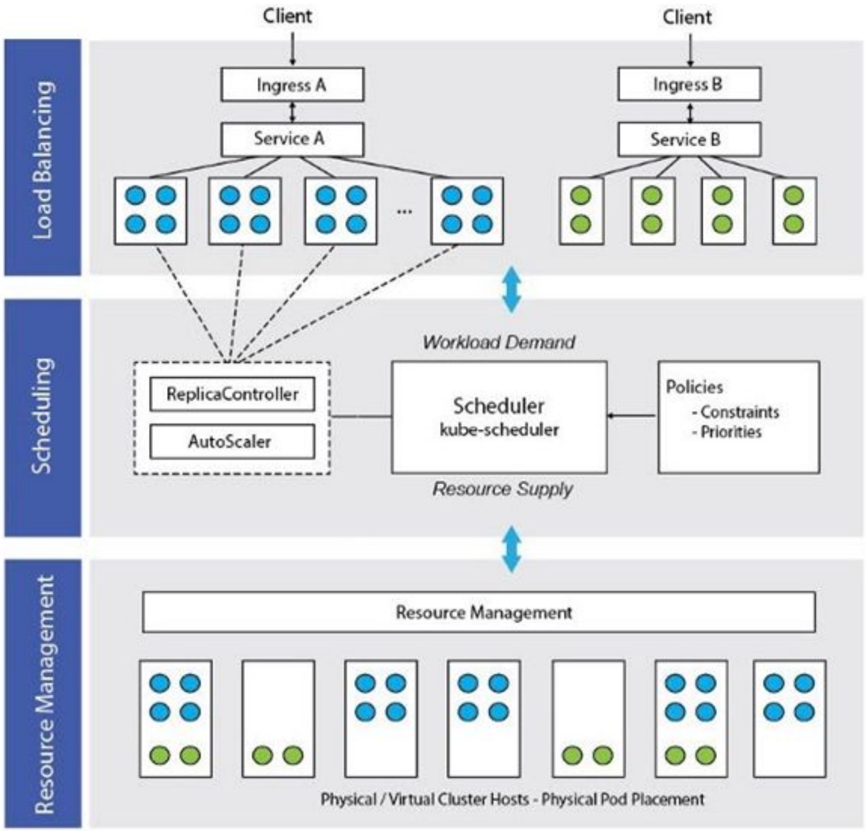


Need for Kubernetes



https://devopscube.com/docker-container-clustering-tools/

Capabilities of Kubernetes



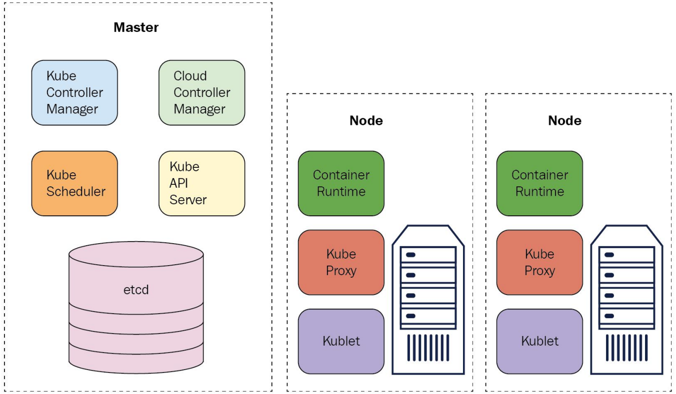
Capabilities of Kubernetes

https://www.guru99.com/kubernetes-tutorial.html

- **Resource Management** is about the efficient allocation of infrastructure resources including CPU and memory.
- **Scheduling** is the process by which pods are matched to available resources. The scheduler considers resource requirements, resource availability and a variety of other user-provided constraints.
- **Load Balancing** involves spreading application load uniformly across a variable number of cluster nodes such that resources are used efficiently.

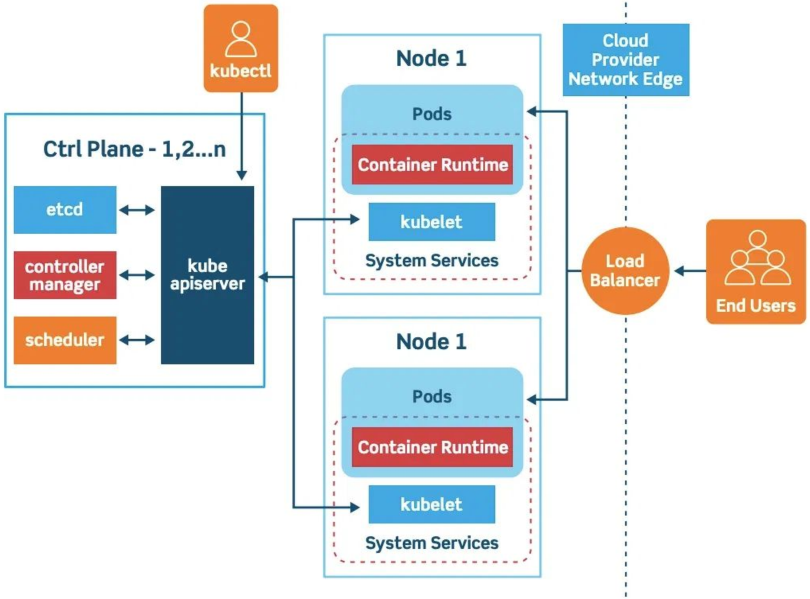
Kubernetes Components and Architecture

From a high level, a Kubernetes environment consists of a **control plane (master)**, a distributed storage system for keeping the cluster state consistent **(etcd)**, and a number of cluster nodes **(Kubelets)**.



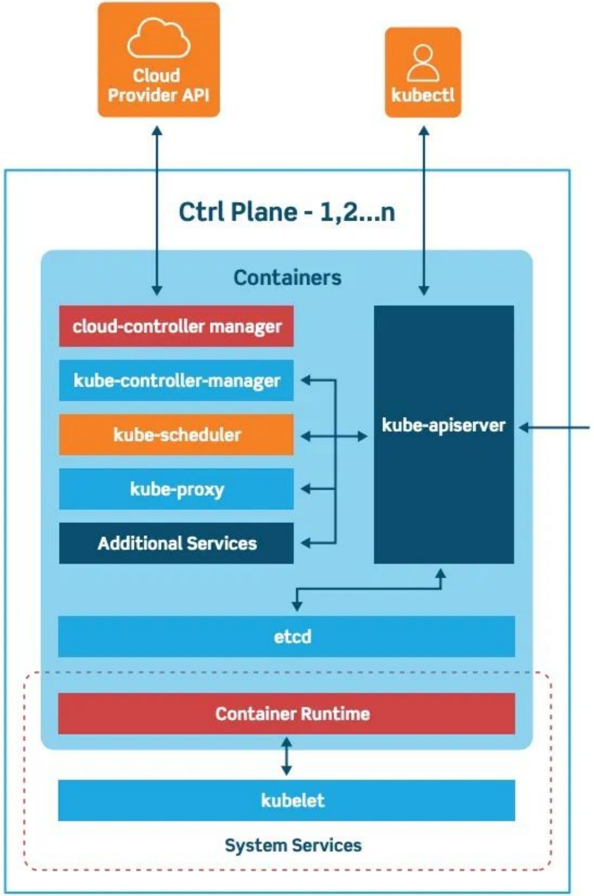
Components of Kubernetes Architecture

https://subscription.packtpub.com/book/cloud_and_networking/9781789805468/1/ch01lvl1sec03/understanding-the-kubernetes-architecture



Architectural overview of Kubernetes

https://www.vamsitalkstech.com/?m=201908



Kubernetes control plane taxonomy

https://www.vamsitalkstech.com/?m=201908

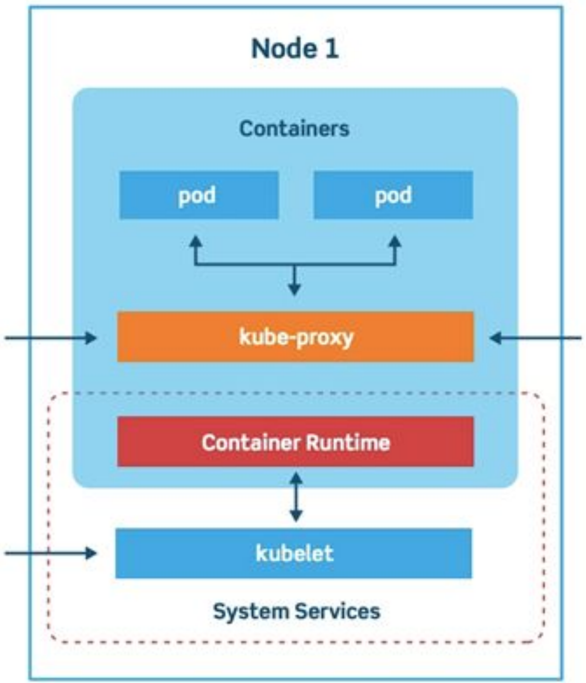
The **control plane** is the system that maintains a record of all Kubernetes objects.

Following are major components of Master Node(Control Plane):

- **kube-apiserver** is the gateway to the cluster, provides APIs to support lifecycle of orchestration (scaling, updates, etc) and also responsible for API validation.
- **etcd** stores the entire configuration and state of the cluster, nodes, pods, and containers.
- **kube-controller-manager** is a daemon that runs the core control loops, watches the state of the cluster, and makes changes to drive status toward the desired state.
- **cloud-controller-manager** is responsible for managing the controllers associated with built in cloud providers
- **kube-scheduler** is responsible for the scheduling of containers across the nodes in the cluster considering resource limitations or guarantees, and affinity and anti-affinity specifications.

Worker Nodes listen to the API Server for new work assignments; they execute the work assignments and then report the results back to the Kubernetes Master node.

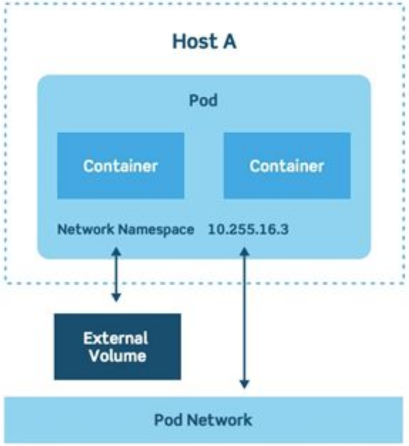
- **kublet** watches for tasks sent from the API Server, executes the task, and reports back to the Master
- **container runtime** pulls images from a container image registry and starts and stops containers. Docker performs this function.
- **kube-proxy** makes sure that each node gets its IP address, implements local iptables and rules to handle routing and traffic load-balancing
- **pod** represents a single instance of an application or running process in Kubernetes, and consists of one or more containers.



Kubernetes worker nodes taxonomy

https://www.vamsitalkstech.com/?m=201908

Pods serve as a 'wrapper' for a single container with the application code. Based on the availability of resources, the Master schedules the pod on a specific node and coordinates with the container runtime to launch the container.



Kubernetes Pod Architecture

https://www.vamsitalkstech.com/?m=201908

The following code is for the Deployment file:

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: mtp-flaskapp-deployment
spec:
  selector:
    matchLabels:
      app: mtp-flaskapp
  replicas: 2
  template:
    metadata:
      labels:
        app: mtp-flaskapp
    spec:
      containers:
      - name: mtp-flaskapp
        image: mtp_geodata_api:latest
        imagePullPolicy: Never
        ports:
          - containerPort: 5000
```

Command for enabling ingress:

```
g193310022@cloudshell:~/mtp-app (mtp-app-318619)$ minikube addons enable ingress
  - Using image docker.io/jettech/kube-webhook-certgen:v1.5.1
  - Using image docker.io/jettech/kube-webhook-certgen:v1.5.1
  - Using image k8s.gcr.io/ingress-nginx/controller:v0.44.0
* Verifying ingress addon...
* The 'ingress' addon is enabled
```

The following code is for ingress.yaml:

```yaml
apiVersion: networking.k8s.io/v1beta1
kind: Ingress
metadata:
  name: mtp-flaskapp-ingress
  annotations:
    kubernetes.io/ingress.class: nginx
    nginx.ingress.kubernetes.io/backend-protocol: "HTTP"
spec:
  rules:
  - http:
      paths:
        -  backend:
             serviceName: mtp-flaskapp-service
             servicePort: http
```

Command for deploying docker image:

```
g193310022@cloudshell:~/mtp-app (mtp-app-318619)$ kubectl apply -f deployment.yaml
deployment.apps/mtp-flaskapp-deployment created
service/mtp-flaskapp-service created
```

Command for viewing nodes:

```
g193310022@cloudshell:~/mtp-app (mtp-app-318619)$ kubectl get nodes
NAME       STATUS   ROLES                  AGE   VERSION
minikube   Ready    control-plane,master   40m   v1.20.7
```

Command for viewing pods:

```
g193310022@cloudshell:~/mtp-app (mtp-app-318619)$ kubectl get pods
NAME                            READY   STATUS    RESTARTS   AGE
mtp-flaskapp-84d798684c-wgrwn   1/1     Running   0          33m
mtp-flaskapp-84d798684c-wshl2   1/1     Running   0          33m
```

Command for scaling pods:

```
g193310022@cloudshell:~/mtp-app (mtp-app-318619)$ kubectl scale deployment mtp-flaskapp --replicas=3
g193310022@cloudshell:~/mtp-app (mtp-app-318619)$ kubectl get deployments mtp-flaskapp
NAME           READY       UP-TO_DATE      AVAILABLE      AGE
mtp-flaskapp   3/3         3               3              11m
```

Command for viewing services:

```
g193310022@cloudshell:~/mtp-app (mtp-app-318619)$ kubectl get services
NAME                   TYPE        CLUSTER-IP     EXTERNAL-IP   PORT(S)    AGE
kubernetes             ClusterIP   10.96.0.1      <none>        443/TCP    42m
mtp-flaskapp-service   ClusterIP   10.110.14.52   5000          5000/TCP   33m
```

Developed Platform Demo

https://github.com/FarheenB/mtp-app.git

Tools and Technologies

- **Google Cloud Platform** - For Development shell
- **Python 3.8** - The scripting language
- **Flask** -The Python framework for creating APIs
- **SQLAlchemy** - The Object Relational Mapper(ORM)
- **SQLite** - The Database
- **ReactJS** - A JavaScript framework
- **HTML** - For front-end
- **CSS** - For interactive UIs
- **Docker** - To form light weight image of the application
- **Kubernetes** - The Container Orchestrating tool
- **Ubuntu:18.04** - The Operating System

Future Scope

- The application aims to run and analyse Big Geospatial datasets and perform Satellite Image Processing. The application is currently able to create and configure data processing projects.
- Move from relational Database SQLite to a document oriented database MongoDB to make querying from base faster.
- Enhancement can be done on scripts added to solve complicated problems on geospatial data cube.
- More algorithms can be added to perform satellite image pre-processing, image corrections and data filtering operations.
- The pre-modeled scripts can be used to perform various case studies such as disaster management, change detection, afforestation etc.

Summary and Conclusion

The present work mainly deals with the development of the architecture of the **cloud platform**. The capabilities of the platform to run geospatial data scripts is explored for SaaS and PaaS. The usage of containers in the proposed architecture increased the level of portability, efficiency, consistency, flexibility and reusability. The containers deploy a separate application for every geospatial data analysis done by the user. Thus the execution of every script in a container is isolated from another. The content of the container includes a Flask Application and Jupyter Notebook along with all the web services needed to store, process and compute the algorithm.

References

K. VANI, "Satellite image processing," 2017 Fourth International Conference on Signal Processing,Communication and Networking (ICSCN), 2017, pp. 1-3, doi: 10.1109/ICSCN.2017.8085410.

SWARNALATHA, P. and PRABU SEVUGAN, editors. Big Data Analytics for Satellite Image Processing and Remote Sensing. IGI Global, 2018. http://doi:10.4018/978-1-5225-3643-7

CHAOWEI YANG, QUNYING HUANG, ZHENLONG LI and KAI LIU AND FEI HU, “Big Data and cloud computing: innovation opportunities and challenges”, 2017, pp. 13-53, doi: 10.1080/17538947.2016.1239771

YAO, XIAOCHUANG & LI, GUOQING & XIA, JUNSHI & BEN, JIN & CAO, QIANQIAN & LONG, ZHAO & YUE, MA & ZHANG, LIANCHONG & ZHU, DEHAI. (2019). Enabling the Big Earth Observation Data via Cloud Computing and DGGS: Opportunities and Challenges. Remote Sensing. 12. 62. 10.3390/rs12010062.

JOSHUA GORE, STEFAN PETERS and DELENE WEBER. (2021) A participatory trail web map based on open source technologies. International Journal of Cartography 0:0, pages 1-20.

Thank You!