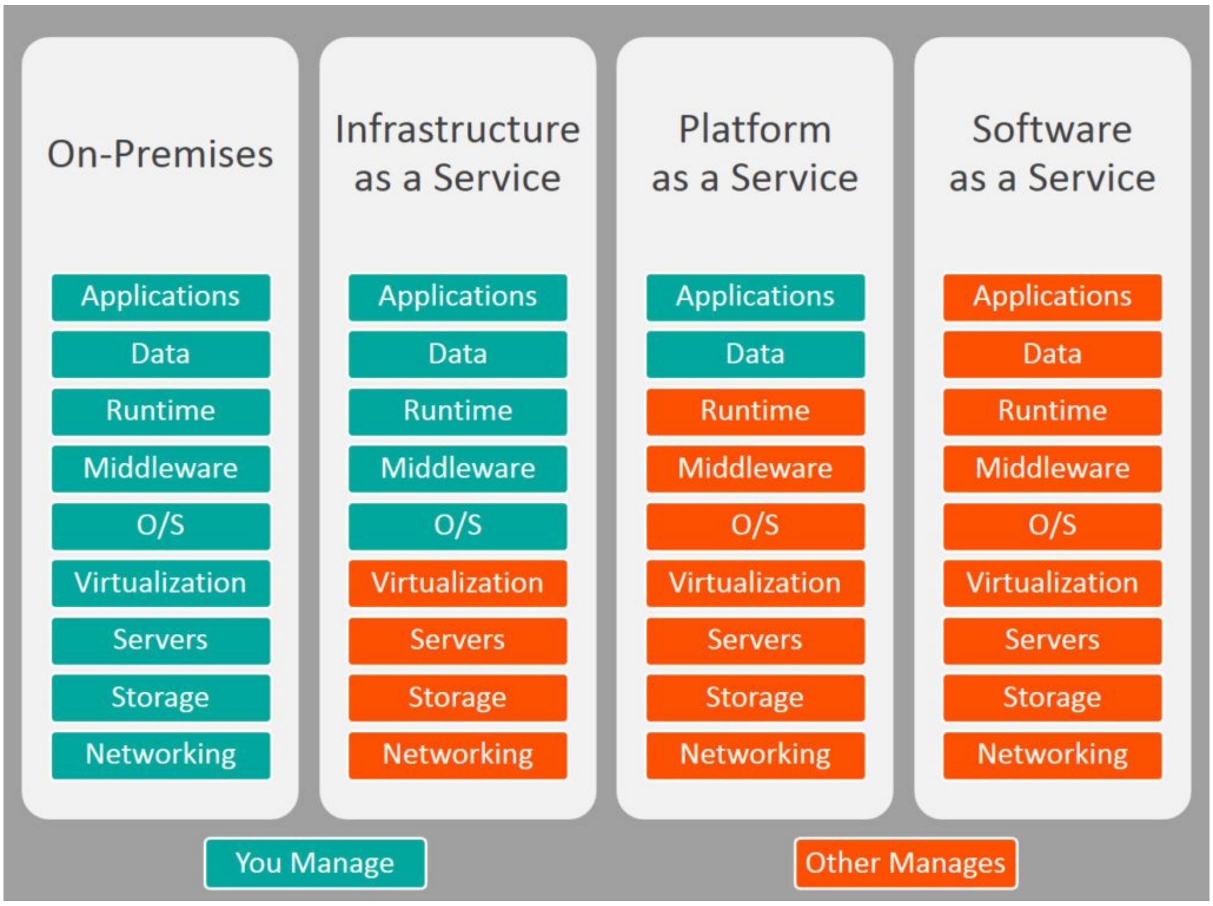
Task 1: Describe/Explain the following:

a. *<u>IaaS, PaaS & SaaS</u>*

IaaS: Infrastructure as a Service
PaaS: Platform as a Service
SaaS: Software as a Service (hinst: Nist)



IaaS
The capability provided to the consumer is to provision processing, storage, networks, and other fundamental computing resources where the consumer is able to deploy and run arbitrary software, which can include operating systems and applications. The consumer does not manage or control the underlying cloud infrastructure but has control over operating systems, storage, and deployed applications; and possibly limited control of select networking components (e.g., host firewalls).

IaaS Delivery
IaaS delivers cloud computing infrastructure, including servers, network, operating systems, and storage, through virtualization technology. IaaS clients complete control over the entire infrastructure through a dashboard or an API.

IaaS provides the same technologies and capabilities as a traditional data center without having to physically maintain or manage all of it.

IaaS Advantages
IaaS offers many advantages, including:

The most flexible cloud computing model
Easy to automate deployment of storage, networking, servers, and processing power
Hardware purchases can be based on consumption
Clients retain complete control of their infrastructure
Resources can be purchased as-needed
Highly scalable

When to Use IaaS
Startups and small companies may prefer IaaS to avoid spending time and money on purchasing and creating hardware and software.
Larger companies may prefer to retain complete control over their applications and infrastructure, but they want to purchase only what they actually consume or need.
Companies experiencing rapid growth like the scalability of IaaS, and they can change out specific hardware and software easily as their needs evolve.

Examples of IaaS
Popular examples of IaaS include DigitalOcean, Linode, Rackspace, Amazon Web Services (AWS), Cisco Metacloud, Microsoft Azure, and Google Compute Engine (GCE).

PaaS

Platform as a Service (PaaS). The capability provided to the consumer is to deploy onto the cloud infrastructure consumer-created or acquired applications created using programming languages, libraries, services, and tools supported by the provider. The consumer does not manage or control the underlying cloud infrastructure including network, servers, operating systems, or storage, but has control over the deployed applications and possibly configuration settings for the application-hosting environment.

PaaS Delivery

PaaS allows businesses to design and create applications that are built into the PaaS with special software components. These applications, sometimes called middleware, are scalable and highly available as they take on certain cloud characteristics.

PaaS Advantages

Simple, cost-effective development and deployment of apps
Scalable
Highly available

Developers can customize apps without the headache of maintaining the software
Significant reduction in the amount of coding needed
Automation of business policy
Easy migration to the hybrid model

When to Use PaaS
Utilizing PaaS is beneficial, sometimes even necessary, in several situations. For example, PaaS can streamline workflows when multiple developers are working on the same development project. If other vendors must be included, PaaS can provide great speed and flexibility to the entire process. PaaS is particularly beneficial if you need to create customized applications. This cloud service also can greatly reduce costs and it can simplify some challenges that come up if you are rapidly developing or deploying an app.

Examples of PaaS
Popular examples of PaaS include AWS Elastic Beanstalk, Windows Azure, Heroku, Force.com, Google App Engine, and OpenShift

SaaS

Software as a Service (SaaS). The capability provided to the consumer is to use the provider's applications running on a cloud infrastructure. The applications are accessible from various client devices through either a thin client interface, such as a web browser (e.g., web-based email), or a program interface. The consumer does not manage or control the underlying cloud infrastructure including network, servers, operating systems, storage, or even individual application capabilities, with the possible exception of limited user-specific application configuration settings.

SaaS Advantages
Greatly reducing the time and money spent on tedious tasks such as installing, managing, and upgrading software. This frees up plenty of time for technical staff to spend on more pressing matters and issues within the organization.

SaaS Characteristics
Managed from a central location
Hosted on a remote server
Accessible over the internet
Users not responsible for hardware or software updates

When to Use SaaS
SaaS may be the most beneficial option in several situations, including:

Startups or small companies that need to launch ecommerce quickly and don't have time for server issues or software
Short-term projects that require quick, easy, and affordable collaboration
Applications that aren't needed too often, such as tax software

Applications that need both web and mobile access are several popular examples of SaaS, including: Google GSuite (Apps), Dropbox, Salesforce, Cisco WebEx, SAP Concur, and GoToMeeting.

b. *OS-level virtualization*

Operating-system-level virtualization is a server-virtualization method where the kernel of an operating system allows for multiple isolated user-space instances, instead of just one. Such instances, which are sometimes called containers and software containers.

Some of popular implementations are as follows:

chroot
Docker
LXC
LXD
Linux-VServer
OpenVZ
Solaris Containers
FreeBSD jail

c. *cgroups*

Docker Engine on Linux also relies on another technology called control groups (cgroups). A cgroup limits an application to a specific set of resources. Control groups allow Docker Engine to share available hardware resources to containers and optionally enforce limits and constraints. For example, you can limit the memory available, cpu, block i/o, network, freezer,... to a specific container.

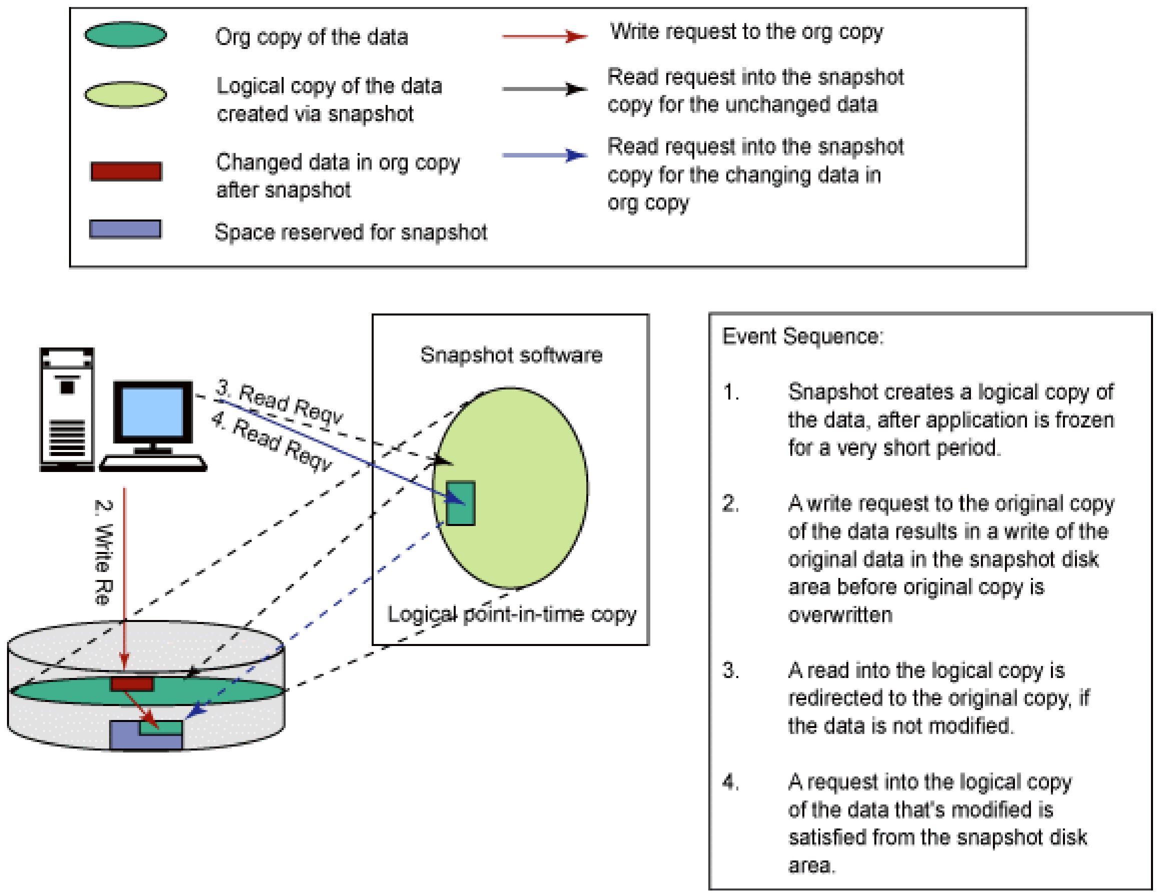
d. *Copy-on-Write (COW) & Snapshots*

Snapshot is a common industry term denoting the ability to record the state of a storage device at any given moment and preserve that snapshot as a guide for restoring the storage device in the event that it fails. A snapshot primarily creates a point-in-time copy of the data. Typically, snapshot copy is done instantly and made available for use by other applications such as data protection, data analysis and reporting, and data replication applications. The original copy of the data continues to be available to the applications without interruption, while the snapshot copy is used to perform other functions on the data.

Snapshots provide an excellent means of data protection. The trend towards using snapshot technology comes from the benefits that snapshots deliver in addressing many of the issues that businesses face. Snapshots enable better application availability, faster recovery, easier back up management of large volumes of data, reduces exposure to data loss, virtual elimination of backup windows, and lowers total cost of ownership (TCO).

A snapshot of a storage volume is created using the pre-designated space for the snapshot. **When the snapshot is first created, only the meta-data about where original data is stored is copied. No physical copy of the data is done at the time the snapshot is created.** Therefore, the creation of the snapshot is almost instantaneous. The snapshot copy then tracks the changing blocks on the original volume as writes to the original volume are performed. The original data that is being written to is copied into the designated storage pool that is set aside for the snapshot before original data is overwritten, hence the name "copy-on-write".
Before a write is allowed to a block, copy-on-write moves the original data block to the snapshot storage. This keeps the snapshot data consistent with the exact time the snapshot was taken. Read requests to the snapshot volume of the unchanged data blocks are redirected to the "copied" blocks in the snapshot, while read requests to active data blocks that have been changed are directed to the original volume. Snapshot contains the meta-data that describes the data blocks that have changed since the snapshot was first created. Note that original data blocks are copied only once into the snapshot storage when the first write request is received.
The following diagram illustrates a snapshot operation that creates a logical copy of the data using copy-on-write method.

Figure 1. Copy-on-write illustration


Org copy of the data
Logical copy of the data created via snapshot
Changed data in org copy after snapshot
Space reserved for snapshot
Write request to the org copy
Read request into the snapshot copy for the unchanged data
Read request into the snapshot copy for the changing data in org copy
3. Read Reqv
4. Read Reqv
2. Write Re
Snapshot software
Logical point-in-time copy

Event Sequence:

1. Snapshot creates a logical copy of the data, after application is frozen for a very short period.
2. A write request to the original copy of the data results in a write of the original data in the snapshot disk area before original copy is overwritten
3. A read into the logical copy is redirected to the original copy, if the data is not modified.
4. A request into the logical copy of the data that's modified is satisfied from the snapshot disk area.

Copy-on-write snapshot might initially impact performance on the original volume while it exists, because write requests to the original volume must wait while original data is being "copied out" to the snapshot. The read requests to snapshot are satisfied from the original volumes if data being read hasn’t changed. However, this method is highly space efficient, because the storage required to create a snapshot is minimal to hold only the data that is changing. Additionally, the snapshot requires original copy of the data to be valid.

e. *High-Availability*

High availability refers to systems that are durable and likely to operate continuously without failure for a long time. The term implies that parts of a system have been fully tested and, in many cases, that there are accommodations for failure in the form of redundant components.

f. *Idempotency*

Idempotence, in programming and mathematics, is a property of some operations such that no matter how many times you execute them, you achieve the same result.

In programming, idempotence can be a property of many different code elements, including functions, methods, requests and statements. Idempotence is a language-agnostic property: It means the same thing in any programming context.

g. *Mutable vs Immutable Infrastructure*

What is mutable infrastructure?
The word “mutable” means “liable to change,” which makes the term “mutable infrastructure” a very apt on.

Traditionally, server architectures have been mutable due to the greater short-term flexibility that the mutable approach provides. However, mutable infrastructure comes at the cost of predictability and consistency between different server deployments, as is possible with immutable infrastructure.

The advantages of mutable infrastructure include:

The infrastructure can more precisely fit the needs of the applications that are running on the server.
Updates are usually faster and can be adapted to each individual server.
Rather than needing to create a new server from scratch (which can seem like a scary prospect), IT staff get to know each server on a “personal” level, which can sometimes help fix problems more quickly.
The drawbacks of mutable infrastructure include:

Technical issues are difficult to diagnose or reproduce because each server has a unique configuration, a phenomenon often known as “configuration drift.”
Changes to the server are not necessarily documented, making version tracking more difficult.
Provisioning servers is usually a long process due to the need for manual configuration.
What is immutable infrastructure?

As you might have guessed, the word “immutable” is the antonym of “mutable,” meaning “unchanging or unable to change.” Immutable infrastructure is IT server infrastructure that, once deployed, cannot be modified. It’s often associated with the software engineering practices DevOps and continuous delivery.

In the event that changes or updates need to be made, an entirely new instance with the proper modifications is deployed onto the server. New environments can be spawned in the cloud in a matter of minutes. This makes immutable infrastructure a much more feasible option for the 96% of businesses that use the cloud.

The advantages of immutable infrastructure include:

Version tracking and rollbacks are much easier. The IT department can keep tabs on each new server or virtual machine as it is deployed.
Tests are easier to run thanks to the consistency in configurations between different servers.
Configuration drift is not possible. If a server is up and running, the IT staff know the exact state of that server and can avoid any unexpected surprises.
The drawbacks of immutable infrastructure include:

The infrastructure is completely unable to be modified in-place. In the event of a zero-day vulnerability, for example, all servers with the same configuration must receive a security update.
The improved agility and dynamism of immutable infrastructure can sometimes be misaligned with traditional IT security practices.

h. *Configuration Management vs. Orchestration*

- Configuration management

Generally, Ansible, Puppet, SaltStack, and Chef are considered to be configuration management (CM) tools and were created to install and manage software on existing server instances (e.g., installation of packages, starting of services, installing scripts or config files on the instance). They do the heavy lifting of making one or many instances perform their roles without the user needing to specify the exact commands. No more manual configuration or ad-hoc scripts are needed.

- Configuration orchestration

Tools like Terraform are considered to be orchestrators. They are designed to provision the server instances themselves, leaving the job of configuring those servers to other tools. Orchestration addresses the requirement to provision environments at a higher level than configuration management. The focus here is on coordinating configuration across complex environments and clusters.

i. *Procedural vs Declarative*

Imperative programming – focuses on how to execute, defines control flow as statements that change a program state.

Examples of programming languages which support the procedural paradigm:
C (and most other legacy languages)
PHP, mostly
In some sense, all major languages

Declarative programming – focuses on what to execute, defines program logic, but not detailed control flow.
Examples of programming languages which support the declarative programming paradigm:
yacc
Treetop
SQL
Regular Expressions
lex
XSLT
markup, troff, CSS, VHDL

j. *Git Sub Module*

Often a code repository will depend upon external code. This external code can be incorporated in a few different ways. The external code can be directly copied and pasted into the main repository. This method has the downside of losing any upstream changes to the external repository. Another method of incorporating external code is through the use of a language's package management system like Ruby Gems or NPM. This method has the downside of requiring installation and version management at all places the origin code is deployed. Both of these suggested incorporation methods do not enable tracking edits and changes to the external repository.

A git submodule is a record within a host git repository that points to a specific commit in another external repository. Submodules are very static and only track specific commits. Submodules do not track git refs or branches and are not automatically updated when the host repository is updated. When adding a submodule to a repository a new .gitmodules file will be created. The .gitmodules file contains meta data about the mapping between the submodule project's URL and local directory. If the host repository has multiple submodules, the .gitmodules file will have an entry for each submodule.

k. *Ansible: Inventory File*

The Ansible inventory file defines the hosts and groups of hosts upon which commands, modules, and tasks in a playbook operate. The file can be in one of many formats depending on your Ansible environment and plugins. The default location for the inventory file is /etc/ansible/hosts. If necessary, you can also create project-specific inventory files in alternate locations.

Ansible: Playbook
An Ansible playbook is an organized unit of scripts that defines work for a server configuration managed by the automation tool Ansible.

l. *Compare Stateful and Stateless Application*

A stateless application is one which depends on no persistent storage. The only thing your cluster is responsible for is the code, and other static content, being hosted on it. That's it, no changing databases, no writes and no left over files when the pod is deleted.
Different pods all across the cluster can work independently with multiple requests coming to them simultaneously. If something goes wrong, you can just restart the application and it will go back to the initial state with little downtime.

A stateful application, on the other hand, has several other parameters it is supposed to look after in the cluster. There are dynamic databases which, even when the app is offline or deleted, persist on the disk. Any stateful service (or application), being run on a Kubernetes cluster, needs to have a balance between: consistency, availability and network partitioning.
Consistency, in the context of a Kubernetes cluster, means every read receives the most recent write or an error message.
But this cuts against availability, one of the most important reasons for having a distributed system. Availability implies that your application functions as close to perfection as possbile, around the clock, with as little error as possible.
You need to have a decentralized way of storing persistent data in a cluster. Commonly referred to as network partitioning. Moreover, your cluster must be able to survive the failure of nodes running the stateful application.

m. *ReplicaSets,Deployments, Pods & Services*

- Pods:

A Pod is the basic execution unit of a Kubernetes application–the smallest and simplest unit in the Kubernetes object model that you create or deploy. A Pod represents processes running on your Cluster.

A Pod encapsulates an application's container (or, in some cases, multiple containers), storage resources, a unique network IP, and options that govern how the container(s) should run. A Pod represents a unit of deployment: a single instance of an application in Kubernetes, which might consist of either a single container or a small number of containers that are tightly coupled and that share resources.

Docker is the most common container runtime used in a Kubernetes Pod, but Pods support other container runtimes as well.

Pods in a Kubernetes cluster can be used in two main ways: Pods that run a single container and Pods that run multiple containers that need to work together.

Each Pod is meant to run a single instance of a given application. If you want to scale your application horizontally (e.g., run multiple instances), you should use multiple Pods, one for each instance. In Kubernetes, this is generally referred to as replication. Replicated Pods are usually created and managed as a group by an abstraction called a Controller.

- Services:

In Kubernetes, a Service is an abstraction which defines a logical set of Pods and a policy by which to access them (sometimes this pattern is called a micro-service). The set of Pods targeted by a Service is usually determined by a selector (see below for why you might want a Service without a selector).

For example, consider a stateless image-processing backend which is running with 3 replicas. Those replicas are fungible—frontends do not care which backend they use. While the actual Pods that compose the backend set may change, the frontend clients should not need to be aware of that, nor should they need to keep track of the set of backends themselves.

The Service abstraction enables this decoupling.

- ReplicaSets

A ReplicaSet's purpose is to maintain a stable set of replica Pods running at any given time. As such, it is often used to guarantee the availability of a specified number of identical Pods.

- Deployments

A Deployment provides declarative updates for Pods and ReplicaSets.
You describe a desired state in a Deployment, and the Deployment Controller changes the actual state to the desired state at a controlled rate. You can define Deployments to create new ReplicaSets, or to remove existing Deployments and adopt all their resources with new Deployments.

Task 2:

Subtask 1:

Clone the project from github (https://github.com/yhl17cmt/Demo1WebApp)

```
$ git clone https://github.com/yhl17cmt/Demo1WebApp
$ cd Demo1WebApp/
$ vim dockerfile
"
FROM maven:3.6.3-ibmjava-8-alpine AS MAVEN_TOOL_CHAIN
COPY pom.xml /tmp/
COPY src /tmp/src/
WORKDIR /tmp/
RUN mvn package

FROM tomcat:latest
COPY --from=MAVEN_TOOL_CHAIN /tmp/target/Demo1WebApp.war
$CATALINA_HOME/webapps/
```

HEALTHCHECK --interval=1m --timeout=3s CMD wget --quiet --tries=1 --spider http://192.168.122.161:8080/Demo1WebApp/ || exit 1
"

Then we run the command:

```
sudo docker image build -t demoweb .
sudo docker run -p 8080:8080 demoweb:latest
```

Question 1: Why might you want to make a multi-stage build ?
Syftet med en multi-stage build är att bygga en slimmad image. Vid en multi-stage build delas bygget upp i flera delsteg (stages) som var och en bygger en image. Detta görs med flera FROM-instruktioner i en och samma Dockerfile. Artefakter från delstegen kan sedan kopieras in i efterföljande images.

<u>*Subtask 2:*</u>
Clone the project from github (https://github.com/yhl17cmt/business-data)

```
$ git clone https://github.com/yhl17cmt/business-data
$ cd business-data/
$ vim dockerfile
```
"

```
FROM mariadb
ENV MYSQL_ALLOW_EMPTY_PASSWORD=true
COPY Northwind_Database.sql /docker-entrypoint-initdb.d/dump.sql
#PORT
EXPOSE 3306
```
"

Then we run the command:

```
docker build -t mariadb .
docker run --rm -it  -v \ $(pwd)/Northwind_Database.sql:/docker-entrypoint-initdb.d/dump.sql
mariadb
```

Question 2: Mounting database dumps like this can be a good solution to use for a test database, but why might you not want to do it like this for a production database?

Because not using password to protect in MySQL will trigger security holes and you will be attacked by hackers.

<u>*Subtask 3:*</u>
Please see file: playbook-subtask3.yml attached in folder.

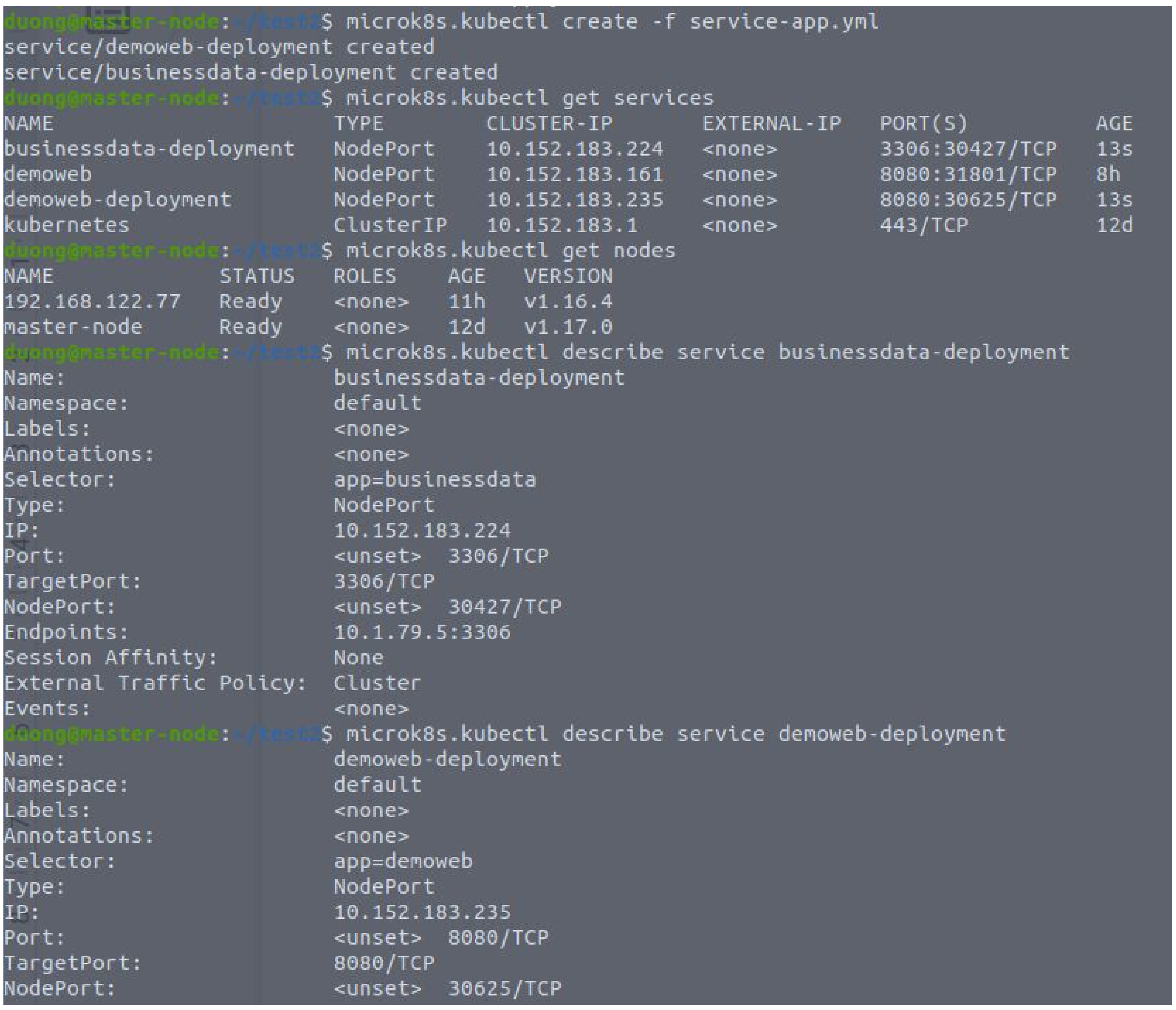
Task 3:

Reading in web page: https://microk8s.io/

Task 4: Define a Deployment equal to T2 and deploy a Pod/ReplicaSet to your cluster and expose it as a Service.

- First I created Service prior to any controllers (like a deployment) so that the Kubernetes scheduler can distribute the pods for the service as they are created by the controller.

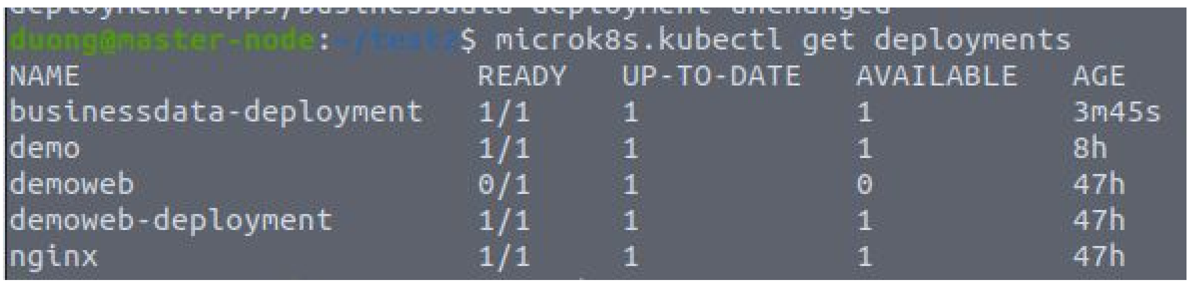
Please check *service-app.yml* in directory.
Then I do this command: microk8s.kubectl create -f service-app.yml
To check the status of services: microk8s.kubectl get services



```
duong@master-node:~/test2$ microk8s.kubectl create -f service-app.yml
service/demoweb-deployment created
service/businessdata-deployment created
duong@master-node:~/test2$ microk8s.kubectl get services
NAME                      TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)          AGE
businessdata-deployment   NodePort    10.152.183.224   <none>        3306:30427/TCP   13s
demoweb                   NodePort    10.152.183.161   <none>        8080:31801/TCP   8h
demoweb-deployment        NodePort    10.152.183.235   <none>        8080:30625/TCP   13s
kubernetes                ClusterIP   10.152.183.1     <none>        443/TCP          12d
duong@master-node:~/test2$ microk8s.kubectl get nodes
NAME             STATUS   ROLES    AGE   VERSION
192.168.122.77   Ready    <none>   11h   v1.16.4
master-node      Ready    <none>   12d   v1.17.0
duong@master-node:~/test2$ microk8s.kubectl describe service businessdata-deployment
Name:                     businessdata-deployment
Namespace:                default
Labels:                   <none>
Annotations:              <none>
Selector:                 app=businessdata
Type:                     NodePort
IP:                       10.152.183.224
Port:                     <unset>  3306/TCP
TargetPort:               3306/TCP
NodePort:                 <unset>  30427/TCP
Endpoints:                10.1.79.5:3306
Session Affinity:         None
External Traffic Policy:  Cluster
Events:                   <none>
duong@master-node:~/test2$ microk8s.kubectl describe service demoweb-deployment
Name:                     demoweb-deployment
Namespace:                default
Labels:                   <none>
Annotations:              <none>
Selector:                 app=demoweb
Type:                     NodePort
IP:                       10.152.183.235
Port:                     <unset>  8080/TCP
TargetPort:               8080/TCP
NodePort:                 <unset>  30625/TCP
```

- Create Deployment yaml file

A deployment is a controller that helps manage the state of my pods. The deployment will define how many pods should be kept up and running with service and which container image should be used.
Please check *deployment_app.yml* in directory
Then I do command: microk8s.kubectl apply -f deployment_app.yml
To check status of deployment: microk8s.kubectl get deployments



```
duong@master-node:~/test2$ microk8s.kubectl get deployments
NAME                       READY   UP-TO-DATE   AVAILABLE   AGE
businessdata-deployment    1/1     1            1           3m45s
demo                       1/1     1            1           8h
demoweb                    0/1     1            0           47h
demoweb-deployment         1/1     1            1           47h
nginx                      1/1     1            1           47h
```

Now i can go to: http://192.168.122.161:30625/Demo1WebApp/ to test this app works.

Task 5:

Please see the attached file: *Microservices.pdf* in folder.