

Docker & Kubernetes

David (Dudu) Zbeda

Linkedin: [linkedin.com/in/davidzbeda](https://www.linkedin.com/in/davidzbeda)

Medium: https://medium.com/@dudu.zbeda_13698

Before you start

- When copying and pasting commands from a presentation into your terminal, some symbols, such as hyphens ("-"), may become corrupted or replaced with incorrect characters.
- For the practice sections, you can use WSL. In the following blog, I explain how to install WSL, Minikube, and Kubectl.
 - https://www.linkedin.com/posts/davidzbeda_wsl2-seamlessly-install-ubuntu-os-docker-activity-7240651600044670976-VCyn?utm_source=share&utm_medium=member_desktop
 - https://medium.com/@dudu.zbeda_13698/wsl2-seamlessly-install-ubuntu-os-docker-and-kubernetes-on-windows-for-a-development-environment-13ce936a275c
- All the files required for the practice sections are available in my repository.
 - <https://github.com/dzbeda/docker-and-Kubernetes-training.git>
- Please note that some parts of the content might seem vague or open to misinterpretation. Keep in mind that this is a presentation, and I'll be explaining everything in detail as I present. So, while reading alone might feel incomplete, the full context will come through when I'm speaking to you! 😊



Docker Agenda



- Tech Evolution
- Docker Basic
- Docker repository
- Docker Practice
- YAML
- Docker Compose

Technology Evolution



5

Once Upon A time

1981 – MS Dos

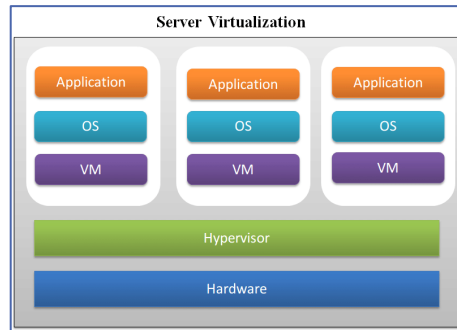
1985 – Windows1

1991 - Unix



1999 – Vmware workstation

2006 – Vmware server



2013 – Docker

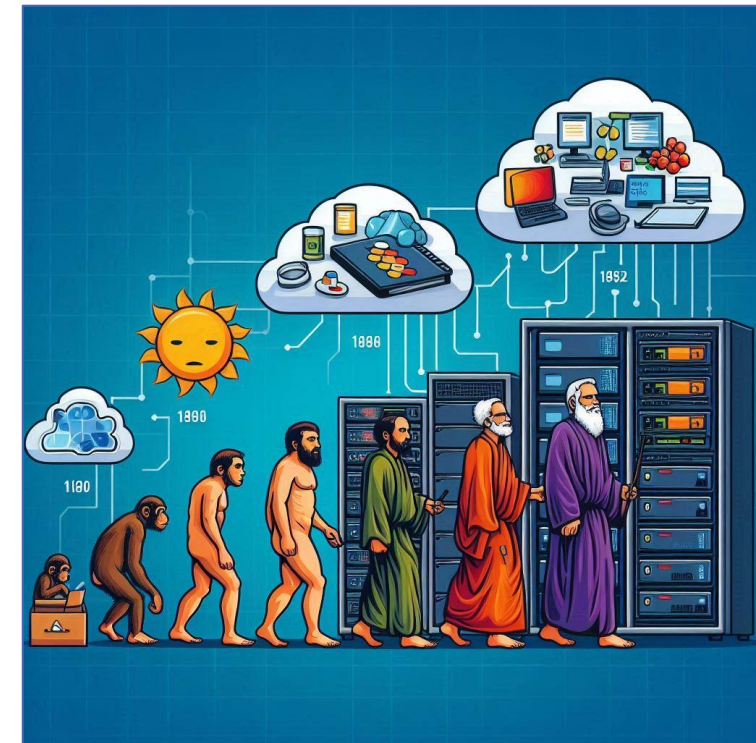
2015 – Docker swarm



2015 - Kubernetes

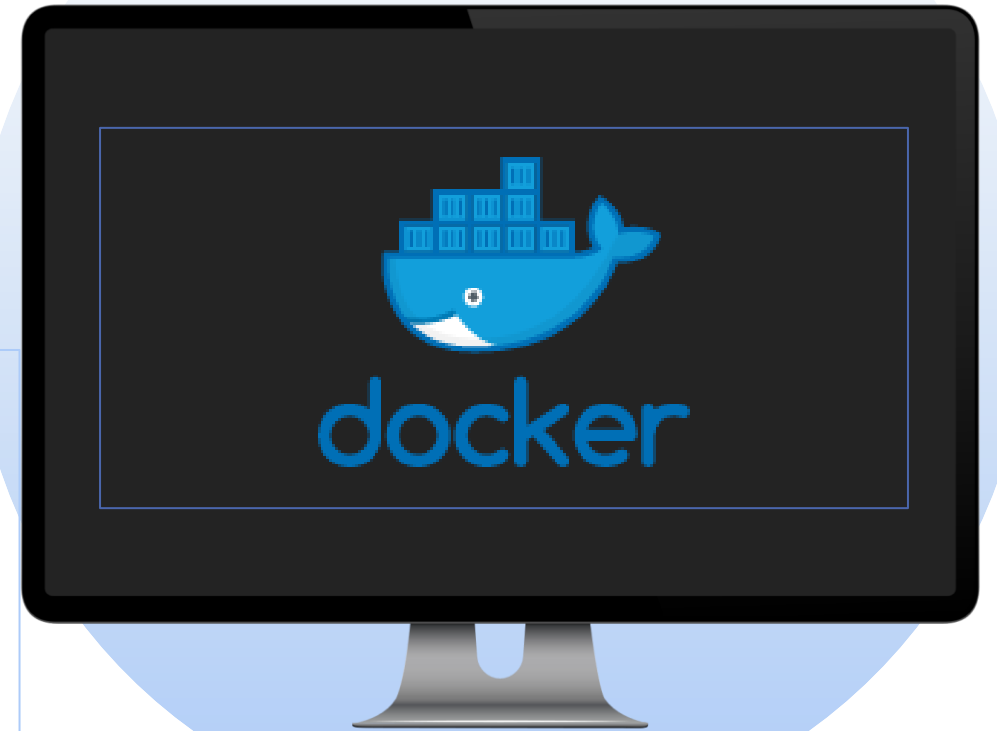


kubernetes



Docker

- The What
- The Why
- The How



Elasticsearch DB What and How

- **What is Elastic Search DB**

- Opensource solution
- Elastic can handle variety of records (Structure , Semi Structure (Json like) and Unstructured data
- No need to define schema
- Data is being index therefore it can be searched
- Elastic is scalable Database , can run on a single node and multi nodes

- **What is needed to install Elastic application**

- Elastic DB requires Linux Distribution or Windows (Lets assume ubuntu)
- Elastic DB is Java based application – So java is needed
- Elastic DB can be scalable – Running more than one single node



elasticsearch

8

Steps for running ElasticDB on physical server

Installation steps:

1. Allocate physical server
2. Install Ubuntu*
3. Install Java
4. Install Single Elastic Node

Steps required after hardware failure:

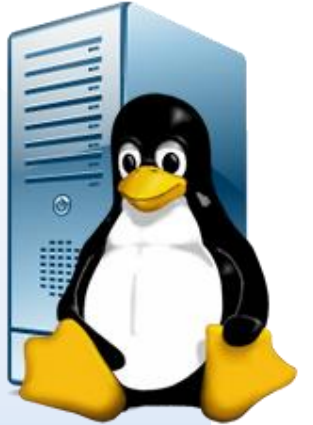
1. Allocate physical server
2. Install Ubuntu*
3. Install Java
4. Install Single Elastic Node

Steps required for scaling :

1. Allocate additional physical server
2. Install Ubuntu*
3. Install Java
4. Install Second Elastic Node instance

*Note:

Need to make sure that the server support the OS release



Steps for running ElasticDB on Virtualization solution

Installation steps:

1. Allocate physical server
2. Install Virtualization Layer
3. Create and Configure Virtual server
4. Install Ubuntu
5. Install Java
6. Install Single Elastic Node

Steps required after hardware failure:

1. Allocate physical server
2. Install Virtualization Layer
3. Create and Configure Virtual server
4. Install Ubuntu
5. Install Java
6. Install Single Elastic Node

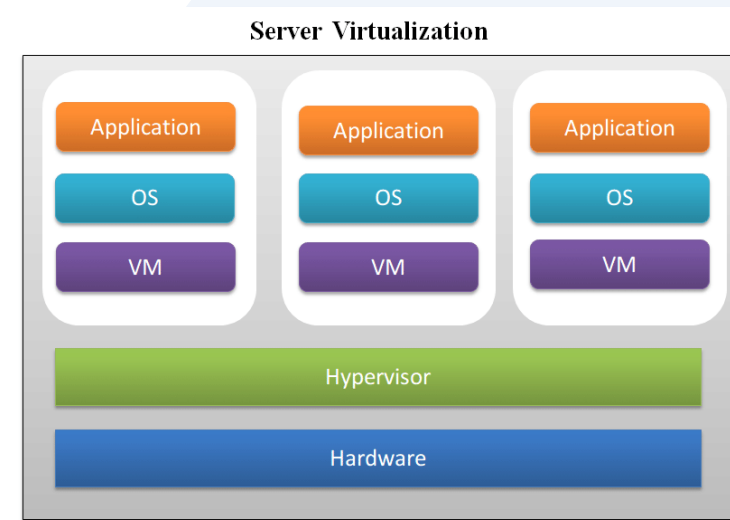


*Note:

- Need to make sure that the server support the OS release
- Running a second Elastic instance on the same physical hardware does not provide protection against hardware failure.

Steps required for scaling :

1. Create additional Virtual server
2. Install Ubuntu
3. Install Java
4. Install Second Elastic Node instance



Steps for running ElasticDB based on Docker

Installation steps:

1. Allocate physical or virtual server
2. Install Ubuntu \ Redhat \ Windows
3. Install Docker engine
4. Start Elastic Docker container



Steps required after hardware failure:

1. Allocate physical or virtual server
Install Ubuntu \ Redhat \ Windows
2. Install Docker engine
3. Start Elastic Docker container

*Note:

- To run Elastic as a container, refer to the README.md file in my repository.
- Running a second Elastic instance on the same physical hardware does not provide protection against hardware failure.

Steps required for scaling :

1. Start second Elastic Docker container

What is Docker & Docker terms

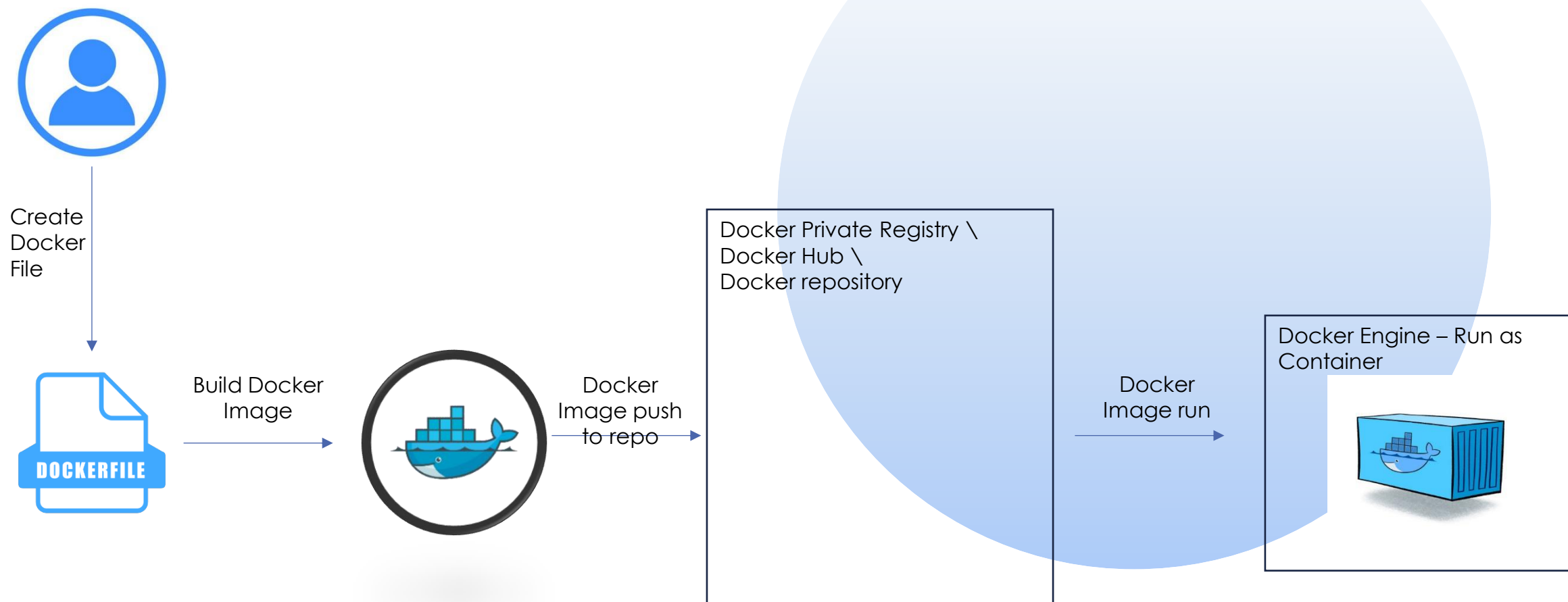
Docker is the open-source technology that helps to containerize apps & automate deployments

- **Containers** - are lightweight *isolated* environments that bundle an application with its dependencies, such as libraries and configurations. They run on a shared OS kernel but provide isolated execution environments for applications.
- **Docker Images** - read-only templates that contain the application code, runtime environment, libraries, and dependencies needed to run a container.
- **Dockerfile** - a script that contains a series of instructions on how to build a Docker image. It specifies the base image, the application code, and any dependencies or configurations needed.
- **Docker Compose** - a tool for defining and running multi-container Docker applications. It uses a YAML file to configure services, networks, and volumes, making it easier to manage complex applications.

What is Docker & Docker terms

- **Docker Engine** - Docker Engine is the runtime that enables containers to run. It consists of a daemon (background service) and a client (command-line interface). The daemon manages containers and images, while the client interacts with the daemon via commands
- **Docker Hub** - Docker Hub is a cloud-based registry service where Docker images are stored and shared. It provides a repository for both public and private images and offers tools for collaboration and versioning. <https://hub.docker.com/>
- **Docker private registry** - Docker images that are saved on Solution \ Company repository .
- **Local Docker repository** - Docker Engine maintains a local repository of images on your host system. When you pull an image from a remote repository (like Docker Hub) or build an image from a Dockerfile, Docker stores these images locally in a directory managed by the Docker Engine
- **Docker Swarm** - Docker Swarm is Docker's native clustering and orchestration tool, which allows users to manage a cluster of Docker engines as a single virtual host. It provides features for scaling, load balancing, and service discovery.

Docker Create and Run flow



Why Docker

- **Portability** - Docker containers encapsulate an application and its dependencies, making it easy to run the same container across different environments (development, testing, production) without compatibility issues.
- **Consistency** - Containers ensure that the application behaves the same way regardless of where it's deployed, reducing "works on my machine" issues
- **Efficiency**- Containers are lightweight compared to virtual machines, as they share the host OS kernel and require fewer resources. This allows for higher density and faster startup times.
- **Scalability**- Docker makes it easy to scale applications horizontally by adding more containers or nodes. Docker **Swarm** and **Kubernetes** can **manage** container **orchestration** and scaling.
- **Isolation** - Containers provide process and resource isolation, which improves security and allows multiple applications to run on the same host without interfering with each other.

Docker Repository



Docker repository

General

- A Docker registry is used to manage Docker images, enabling you to store, distribute, and share container images efficiently
- There are 3 main types of Docker Registry

Local Docker repository –

- **Managed by:** The Docker engine.
- **Location:** Resides on your local server.
- **Usage:** Stores images locally on the machine for development or testing.
- You can list all image by running : > **docker image ls**

Docker Hub repository –

- Cloud Solution for Public and private repository.
- Public images are freely available from the Docker Hub Container Image Library.
- Private repositories allow saving private images, accessible after logging in.
 - Use docker login to authenticate for private repositories
- [Docker Hub Container Image Library | App Containerization](#)

Docker repository

Private Docker repository –

- Typically found in companies or on-premises production environments
- Provides secure and controlled storage for container images.
- Often implemented using third-party solutions such as
 - JFrog Artifactory
 - Nexus Repository Manager
 - Harbor
 - Docker Registry (the open-source Docker solution).



Docker repository

- A Docker image name is defined using the following main parameters:

Format: <repository-url>/<image name>:<tag> <image-ID>

1. <repository-url>

- The URL of the repository from which the image is pulled or to which it is pushed.
- Example: docker.io (Docker Hub's default repository URL) or a custom URL for private registries like registry.example.com.

2. <Image-name>

- The name of the image, typically describing its purpose or contents.
- Example: nginx, my-app, or python.


3. <tag>

- Represents the version of the image.
- If no tag is specified, Docker defaults to the latest tag.
- Example: :1.18 or :latest.

4. <Image-ID>

- A unique identifier for the image.
- This is a truncated version of the SHA-256 hash of the image's contents.
- Example: d64eaf2770d

Interacting with docker registry

- The **Docker Engine CLI** enables interaction with the local Docker registry.
 - List Images : `docker image ls`
 - Pull an image: `docker pull <repository-url>/<image-name>:<tag>`
 - Push an image: `docker push <repository-url>/<image-name>:<tag>`
 - Build an image: `docker build -t <repository-url>/<image-name>:<tag> .`
- 

Docker repository – Login

- Docker engine needs to integrate with a docker registry
 - By default, the integration is done with Docker Hub
- To have the option push images to docker registry that is protected with username and password you will need to use **docker login** command
 - By default, docker login will login to docker.io (docker hub)
- ✓ To login to additional repository run > **docker login <repo-name>**
- ✓ Insecure docker repositories should be defined **/etc/docker/daemon.json**
- ✓ Login credentials are saved in **~/.docker/config.json**

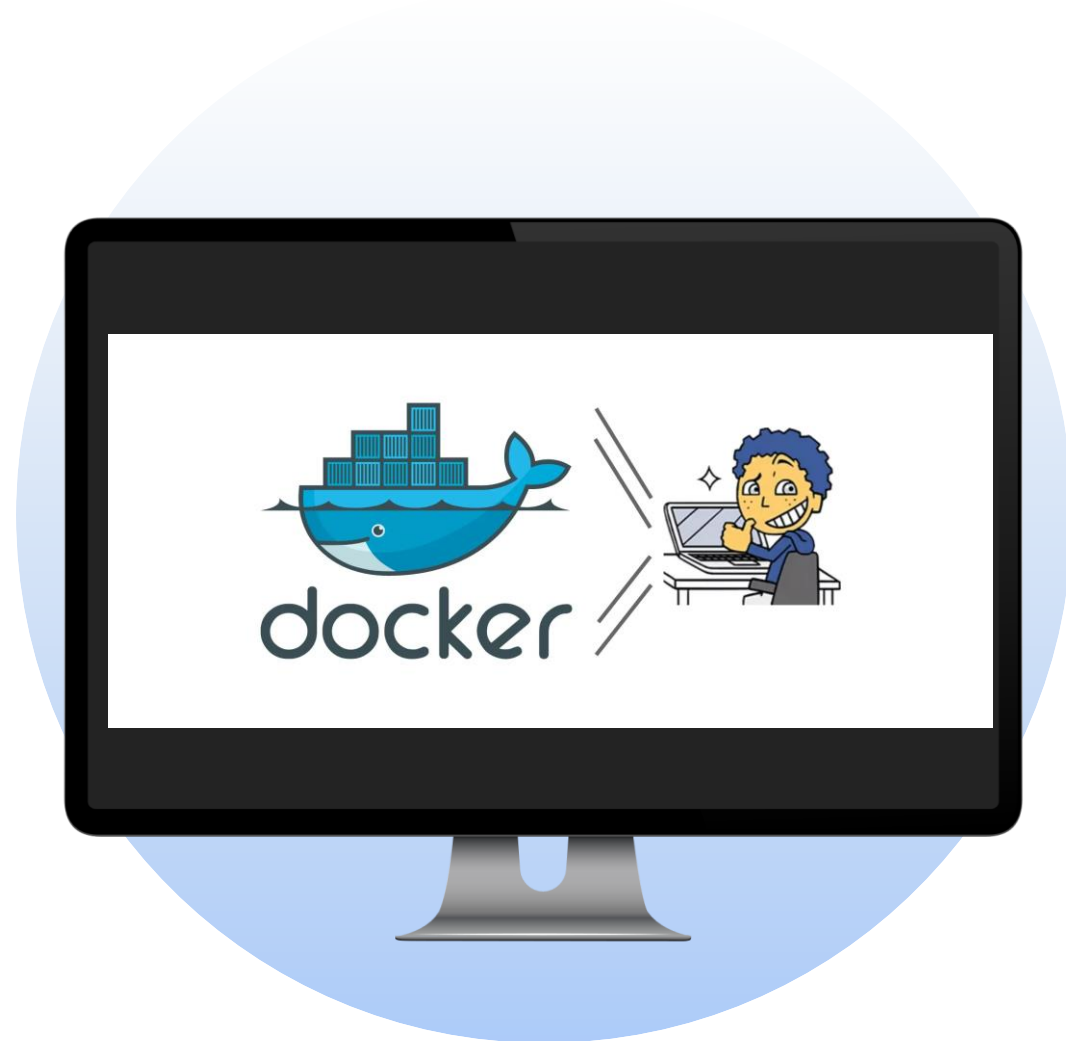


Docker repository – Exercise

- List all image in your **local** docker registry > **docker image ls**
- Pull image from docker Hub > **docker pull nginx**
 - Note: Pulling image from docker HUB dose not requires login



Docker Practice



Let's create our first docker – Docker File & Image build

1. Create new file named Dockerfile
2. Add the following line : FROM ubuntu
3. Save the file
4. Run > **docker build -t class1:v1 .**

What is
FROM ubuntu ?

```
[root@IL-DZBEDA class1]# podman build -t class1:v1 .
WARN[0000] Using cgroups-v1 which is deprecated in favor of cgroups-v2 with Podman v5 and will be removed in a future version. Set environment variable 'PODMAN_IGNORE_CGROUPSV1_WARNING' to hide this warning.
STEP 1/1: FROM ubuntu
Resolved "ubuntu" as an alias (/etc/containers/registries.conf.d/000-shortnames.conf)
Trying to pull docker.io/library/ubuntu:latest...
Getting image source signatures
Copying blob 9c704ecd0c69 done |
Copying config 35a8880255 done |
Writing manifest to image destination
COMMIT class1:v1
--> 35a88802559d
Successfully tagged localhost/class1:v1
Successfully tagged docker.io/library/ubuntu:latest
35a88802559dd2077e584394471ddaa1a2c5bfd16893b829ea57619301eb3908
[root@IL-DZBEDA class1]#
```

Docker From – Base Images

1. Operating System Base Images

These provide a minimal operating system environment.

- FROM ubuntu:latest – For an Ubuntu-based image (you can specify versions like ubuntu:20.04).
- FROM debian:latest – For a Debian-based image.
- FROM alpine:latest – For a very lightweight Linux distribution (often under 10 MB).

2. Language Runtime Base Images

If you're building an app in a specific language, use a language-specific image.

- FROM node:18 – For Node.js applications (you can specify the version).
- FROM python:3.11 – For Python applications.
- FROM golang:1.20 – For Go applications.

3. Application-Specific Base Images

These are for specific tools or services you want to run.

- FROM nginx:alpine – To run the Nginx web server (you can also use nginx:latest).
- FROM mysql:8 – For MySQL database.
- FROM redis:latest – For Redis.

4. Minimal Base Images

These are extremely minimal images often used in highly optimized or security-sensitive environments.

- FROM scratch – An empty base image; you build everything from scratch (often used in conjunction with statically compiled binaries).

Let's create our first docker – Docker File & Image build

The ubuntu image was pulled from docker HUB and new image was created **But where it is saved ?**

1. Run > **docker image ls**

```
[root@IL-DZBEDA class1]# podman images
WARN[0000] Using cgroups-v1 which is deprecated in favor of cgroups-v2 with Podman v5 and will be removed in a future version. Set environment variable 'PODMAN_IGNORE_CGROUPSV1_WARNING' to hide this warning.
REPOSITORY          TAG          IMAGE ID      CREATED       SIZE
localhost/class1    v1          35a88802559d  2 months ago  80.6 MB
docker.io/library/ubuntu latest      35a88802559d  2 months ago  80.6 MB
quay.io/podman/hello latest      5dd467fce50b  2 months ago  787 kB
[root@IL-DZBEDA class1]#
```

2. Run the image as container by running > **docker run -d class1:v1**

3. Run > **docker ps -a** – **We can see that the container was exited – Why ?**

```
[root@IL-DZBEDA class1]# docker ps -a
Emulate Docker CLI using podman. Create /etc/containers/nodocker to quiet msg.
WARN[0000] Using cgroups-v1 which is deprecated in favor of cgroups-v2 with Podman v5 and will be removed in a future version. Set environment variable 'PODMAN_IGNORE_CGROUPSV1_WARNING' to hide this warning.
CONTAINER ID   IMAGE                                COMMAND                  CREATED        STATUS              PORTS          NAMES
6a8a654ead8f   quay.io/podman/hello:latest         /usr/local/bin/po...    8 days ago    Exited (0) 8 days ago                exciting_shtern
1bd025f005dc   quay.io/podman/hello:latest         /usr/local/bin/po...    7 days ago    Exited (0) 7 days ago                zen_lamarr
59cc5fb24915   localhost/class1:v1                 /bin/bash               2 minutes ago Exited (0) 2 minutes ago             boring_hofstadter
```

Let's create our first docker – Docker File & Image build

1. Update the Dockerfile you have created and Add the following lines :
 1. RUN apt update && apt install -y procps
 2. CMD ["sleep", "300"]
2. Save the file
3. Run > **docker build -t class1:v2 .**
4. Run > **docker image ls**

```
[root@IL-DZBEDA class1]# podman images
WARN[0000] Using cgroups-v1 which is deprecated in favor of cgroups-v2 with Podman v5 and will be removed in the future. Set the environment variable 'PODMAN_IGNORE_CGROUPSV1_WARNING' to hide this warning.
REPOSITORY          TAG          IMAGE ID      CREATED       SIZE
localhost/class1    v2           9eb6d13fc841  28 seconds ago  120 MB
localhost/class1    v1           35a88802559d  2 months ago   80.6 MB
docker.io/library/ubuntu latest       35a88802559d  2 months ago   80.6 MB
quay.io/podman/hello latest       5dd467fce50b  2 months ago   787 kB
[root@IL-DZBEDA class1]#
```

5. Run > **docker run -d class1:v2**
6. Run > **docker ps** – now we can see the container that is running – why? when it is going to stop ?

```
[root@IL-DZBEDA ~]# docker ps
Emulate Docker CLI using podman. Create /etc/containers/nodocker to quiet msg.
WARN[0000] Using cgroups-v1 which is deprecated in favor of cgroups-v2 with Podman v5 and will be removed in the future. Set the environment variable 'PODMAN_IGNORE_CGROUPSV1_WARNING' to hide this warning.
CONTAINER ID   IMAGE          COMMAND                  CREATED        STATUS        PORTS        NAMES
6fd16773165f   localhost/class1:v2  sleep 300               3 minutes ago  Up 3 minutes  -           goofy_vaghan
[root@IL-DZBEDA ~]#
```

Let's create our first docker – Docker File & Image build

1. To update the sleep for 30 sec , run the following > `docker run -d class1:v2 sleep 30`

```
[root@IL-DZBEDA class1]# podman run class1:v2 sleep 30
WARN[0000] Using cgroups-v1 which is deprecated in favor of cgroups-v2 with Podman v5 and will be removed in a future version. Set environment variable 'PODMAN_IGNORE_CGROUPSV1_WARNING' to hide this warning.
[root@IL-DZBEDA class1]#
```

```
[root@IL-DZBEDA ~]# docker ps -a
Emulate Docker CLI using podman. Create /etc/containers/nodocker to quiet msg.
WARN[0000] Using cgroups-v1 which is deprecated in favor of cgroups-v2 with Podman v5 and will be removed in a future version. Set environment variable 'PODMAN_IGNORE_CGROUPSV1_WARNING' to hide this warning.
CONTAINER ID   IMAGE                                COMMAND                  CREATED        STATUS              PORTS          NAMES
6a8a654ead8f   quay.io/podman/hello:latest         /usr/local/bin/po...    8 days ago    Exited (0) 8 days ago                exciting_shtern
1bd025f005dc   quay.io/podman/hello:latest         /usr/local/bin/po...    7 days ago    Exited (0) 7 days ago                zen_lamarr
bb666d4fed11   localhost/class1:v2                 sleep 30                 26 seconds ago Up 26 seconds                elastic_hodgkin
[root@IL-DZBEDA ~]# docker ps -a
Emulate Docker CLI using podman. Create /etc/containers/nodocker to quiet msg.
WARN[0000] Using cgroups-v1 which is deprecated in favor of cgroups-v2 with Podman v5 and will be removed in a future version. Set environment variable 'PODMAN_IGNORE_CGROUPSV1_WARNING' to hide this warning.
CONTAINER ID   IMAGE                                COMMAND                  CREATED        STATUS              PORTS          NAMES
6a8a654ead8f   quay.io/podman/hello:latest         /usr/local/bin/po...    8 days ago    Exited (0) 8 days ago                exciting_shtern
1bd025f005dc   quay.io/podman/hello:latest         /usr/local/bin/po...    7 days ago    Exited (0) 7 days ago                zen_lamarr
bb666d4fed11   localhost/class1:v2                 sleep 30                 28 seconds ago Up 29 seconds                elastic_hodgkin
[root@IL-DZBEDA ~]# docker ps -a
Emulate Docker CLI using podman. Create /etc/containers/nodocker to quiet msg.
WARN[0000] Using cgroups-v1 which is deprecated in favor of cgroups-v2 with Podman v5 and will be removed in a future version. Set environment variable 'PODMAN_IGNORE_CGROUPSV1_WARNING' to hide this warning.
CONTAINER ID   IMAGE                                COMMAND                  CREATED        STATUS              PORTS          NAMES
6a8a654ead8f   quay.io/podman/hello:latest         /usr/local/bin/po...    8 days ago    Exited (0) 8 days ago                exciting_shtern
1bd025f005dc   quay.io/podman/hello:latest         /usr/local/bin/po...    7 days ago    Exited (0) 7 days ago                zen_lamarr
bb666d4fed11   localhost/class1:v2                 sleep 30                 32 seconds ago Exited (0) 2 seconds ago            elastic_hodgkin
[root@IL-DZBEDA ~]#
```

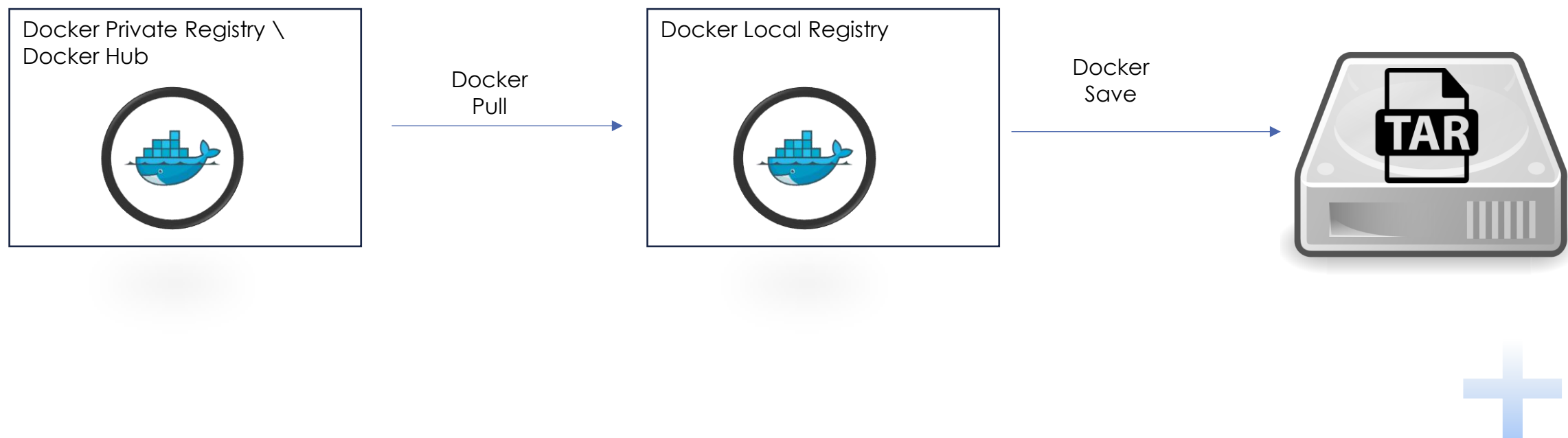


Saving and uploading a docker image file

1. Login to docker hub and locate the Nginx image
2. Pull the docker image from Docker Hub to your local docker repository by running > **docker pull nginx:latest**
3. Run "docker images" to see the images in your local docker registry
4. Save the docker image to a tar file by running > **docker save -o class1-nginx.tar <docker-image-name>:<tag>**
5. Copy the tar file to your instance running the docker engine
6. Load the docker to your local repository by running > **docker load -i class1-nginx.tar**
7. Run "docker image ls" to check that the image have been save on your local docker repository



Docker Save



Docker Load



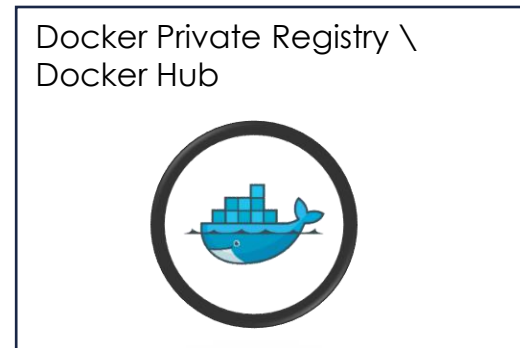
Docker
Load



Docker
Tag



Docker
Push



NGINX Docker

The goal for this exercise

- Upload docker image to you local docker repository
- Run Nginx container using command line
 - Mount the Welcome volume to local disk
 - Connect to NGINX
- Create the Same with Docker File

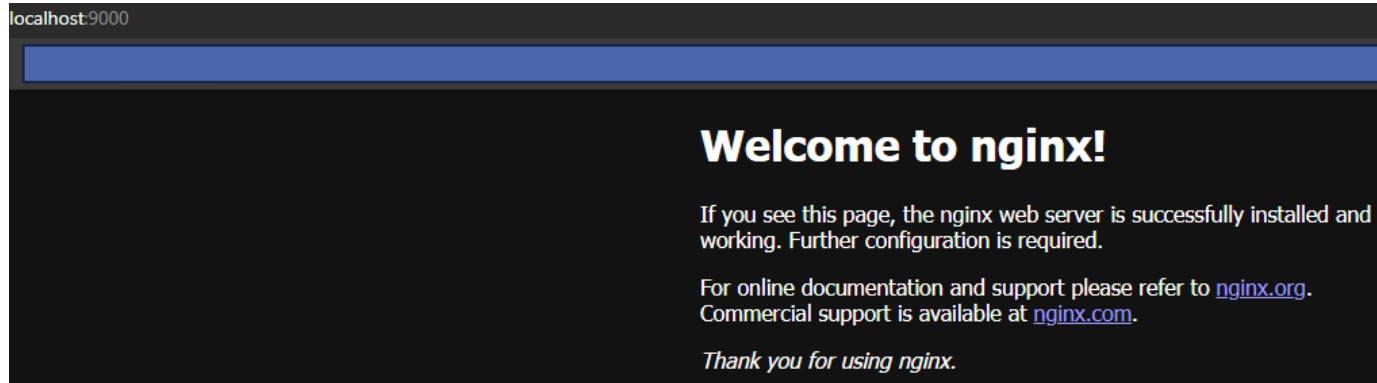
Exercise steps – Instructor

1. Run nginx container by running > `docker run -d - -name class1-nginx-example -p 9000:80 nginx:latest`
2. Run > `docker ps`

```
root@IL-DZBEDA:/mnt/c/Users/dzbeda/Desktop/Traning# docker ps
CONTAINER ID   IMAGE          COMMAND                  CREATED        STATUS        PORTS
NAMES
c8b92928ab75   nginx:latest   "/docker-entrypoint..."   58 seconds ago   Up 58 seconds   0.0.0.0:9000->80/tcp, :::9000->80/tcp   class1-nginx-example
root@IL-DZBEDA:/mnt/c/Users/dzbeda/Desktop/Traning#
```

NGINX Docker

3. Open the browser <http://localhost:9000>



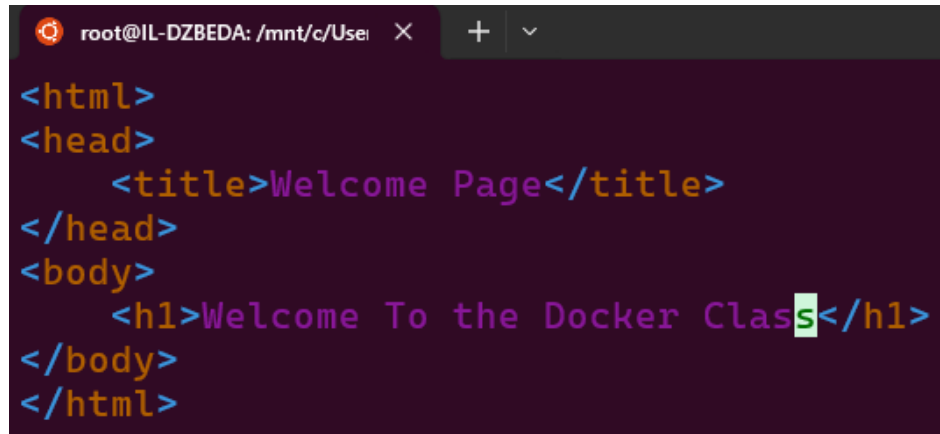
NGINX Docker - Logs

1. Run > **docker ps** and located the container ID
2. Run the following command to check the docker logs > **docker logs -f <container-id>**
3. Refresh the page and check the logs
4. Stop the container by running > **docker stop <container-id>**
5. Run again the command > **docker run -d - -name class1-nginx-example -p 9000:80 nginx:latest**
What happened and why ?
6. Delete the container by running > **docker rm class1-nginx-example**
7. Run > **docker run -d --rm --name class1-nginx-example -p 9000:80 nginx:latest**
8. Stop the container by running > **docker stop <container-name>**
9. Run > **docker ps -a** - Can you see the container ?

NGINX Docker – Update Login message

Create custom docker image

1. Create a new file named index.html
 1. Include the following content and update the welcome message as you wish

A screenshot of a terminal window with a dark background. The terminal shows the creation of an index.html file with the following HTML content:

```
<html>
<head>
  <title>Welcome Page</title>
</head>
<body>
  <h1>Welcome To the Docker Class</h1>
</body>
</html>
```

 The terminal window has a title bar that reads "root@IL-DZBEDA: /mnt/c/Usei" and includes standard window controls (close, maximize, minimize).

2. Save the file

NGINX Docker – Copy file

Create s custom docker image

1. Create new file by running > **vi Dockerfile-nginx-custom** and include the following content

```
# Use the official Nginx image as the base image
FROM nginx:latest

# Copy the custom index.html to the default Nginx location
COPY index.html /usr/share/nginx/html/index.html
```

2. Create new image buy running > **docker build -f Dockerfile-nginx-custom -t class1-nginx-custom:v1 .**
3. Run docker image – Can you explain what we did?
4. Run the command > **docker run -d - -name class1-nginx-example -p 9000:80 class1-nginx-custom:v1**
5. Open the browser <http://localhost:9000>



NGINX Docker – Copy file

Connect to your container

1. Run `> docker ps` and locate the container-id
2. Connect to the container by running the command `> docker exec -it <container-id> sh`
3. Run “cd /usr/share/nginx/html/”
4. Run `> ls`
5. Run `> more index.html`
6. Run Exit in order to exit the interactive mode
7. Stop the container using `> docker ps` command



NGINX Docker – Mount Volume

Let Improve the solution

1. Copy the index.html in your computer under **c:\html** folder
2. Update the index.html with new message
3. Run the container by running the command: **> docker run -d - -name class1-nginx-example2 -p 9000:80 -v /mnt/c/html:/usr/share/nginx/html nginx:latest**
4. Open the browser <http://localhost:9000>
5. Update the index.html file on you computer and refresh your browser

NGINX Docker – Upload to Docker HUB

Let save our image in docker hub under our private location

1. Run > **sudo docker login docker.io**
 1. Enter your email\username and password
2. Run **docker image ls** and located your customize image
3. Update your custom image tag to include the repos name by running > **docker tag class1-nginx-custom:v1 docker.io/<user-name-in-docker-hub>/<new-image-name>:<tag>**
4. Push the docker by running > **docker push docker.io/<user-name-in-docker-hub>/<new-image-name>:<tag>**
5. Check your private repo in Docker Hub

Docker Network

- When installing a docker engine new interface card will be created & and Bridge network is created by default

- Run > **ip a**

```
valid_lft forever preferred_lft forever
4: docker0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue state DOWN group default
    link/ether 02:42:d4:71:b0:67 brd ff:ff:ff:ff:ff:ff
    inet 172.17.0.1/16 brd 172.17.255.255 scope global docker0
        valid_lft forever preferred_lft forever
    inet6 fe80::42:d4ff:fe71:b067/64 scope link
        valid_lft forever preferred_lft forever
```

- Run > **docker network ls** to check available networks

```
root@IL-DZBEDA:/mnt/c/devops/k8s-scan# docker network ls
NETWORK ID          NAME       DRIVER  SCOPE
0fdf4061d675        bridge    bridge  local
e07a4fa63850        host      host     local
4f599cac2e8d        none      null     local
```



Docker Network

- When running a container, it will be assigned by default to the bridge network , and additional network instance will be created in the host level

- Run > **ip a**

```
4: docker0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default
    link/ether 02:42:d4:71:b0:67 brd ff:ff:ff:ff:ff:ff
    inet 172.17.0.1/16 brd 172.17.255.255 scope global docker0
        valid_lft forever preferred_lft forever
    inet6 fe80::42:d4ff:fe71:b067/64 scope link
        valid_lft forever preferred_lft forever
14: vethe6e2a67@if13: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue master docker0 state UP group default
    link/ether b2:fd:03:ce:c4:38 brd ff:ff:ff:ff:ff:ff link-netnsid 0
    inet6 fe80::b0fd:3ff:fece:c438/64 scope link
        valid_lft forever preferred_lft forever
```

- Run > **docker network inspect bridge** to check which containers are assigned to the network
- Run > **bridge link**

- Additional information can be found in the following link:

<https://youtu.be/bKFMS5C4CG0?si=em7FxWhvoTvyznBI>

41 Docker Restart policy

➤ When running container, you can choose between different restart policies

Restart Policy	Description	Restarts After Failure	Restarts After Server Reboot
no	Default. The container will not restart automatically, even after a failure or system reboot.	No	No
always	The container will always restart, regardless of the exit status or system reboot.	Yes	Yes
on-failure	The container will restart only if it exits with a non-zero exit code (failure).	Yes	No
unless-stopped	The container will always restart unless it is manually stopped. It will restart after reboots.	Yes	Yes

✓ To run container with restart option , run > **docker run - -restart=always nginx**

YAML



YAML - YML

- **Definition:** YAML is a versatile, human-readable data serialization language commonly used for writing configuration files.
The presentation of data is based on Key-Value pair
- **Key Features:**
 - **Human-Readable:** Easy to read and write for humans, making it popular for configuration files.
 - **Data Structures:** Supports complex data structures like lists , Scalars & dictionaries.
 - **Indentation-Based:** Uses indentation to define structure, similar to Python.
 - **YAML is case-sensitive**
 - **Empty lines are ignored.**
- **Common Uses:**
 - **Configuration Files:** Widely used in applications, frameworks, and tools (e.g., Docker Compose, Kubernetes).
 - **Data Serialization:** Transferring data between different programming languages.
 - **Templates:** Used in configuration management tools like Ansible.

Note: Do not use TAB spacing in YAML !!!

44 YAML - YML - Examples

Basic Key-Value Pair:

```
yaml
name: John Doe
age: 30
```

You can create a list by starting each item with a dash followed by a space:

```
yaml
fruits:
  - apple
  - banana
  - orange
```

Nested Lists:

YAML also supports nested lists:

```
yaml
grocery:
  fruits:
    - apple
    - banana
  vegetables:
    - carrot
    - lettuce
```

Nested Key-Value Pairs:

You can also nest key-value pairs within other key-value pairs (mappings):

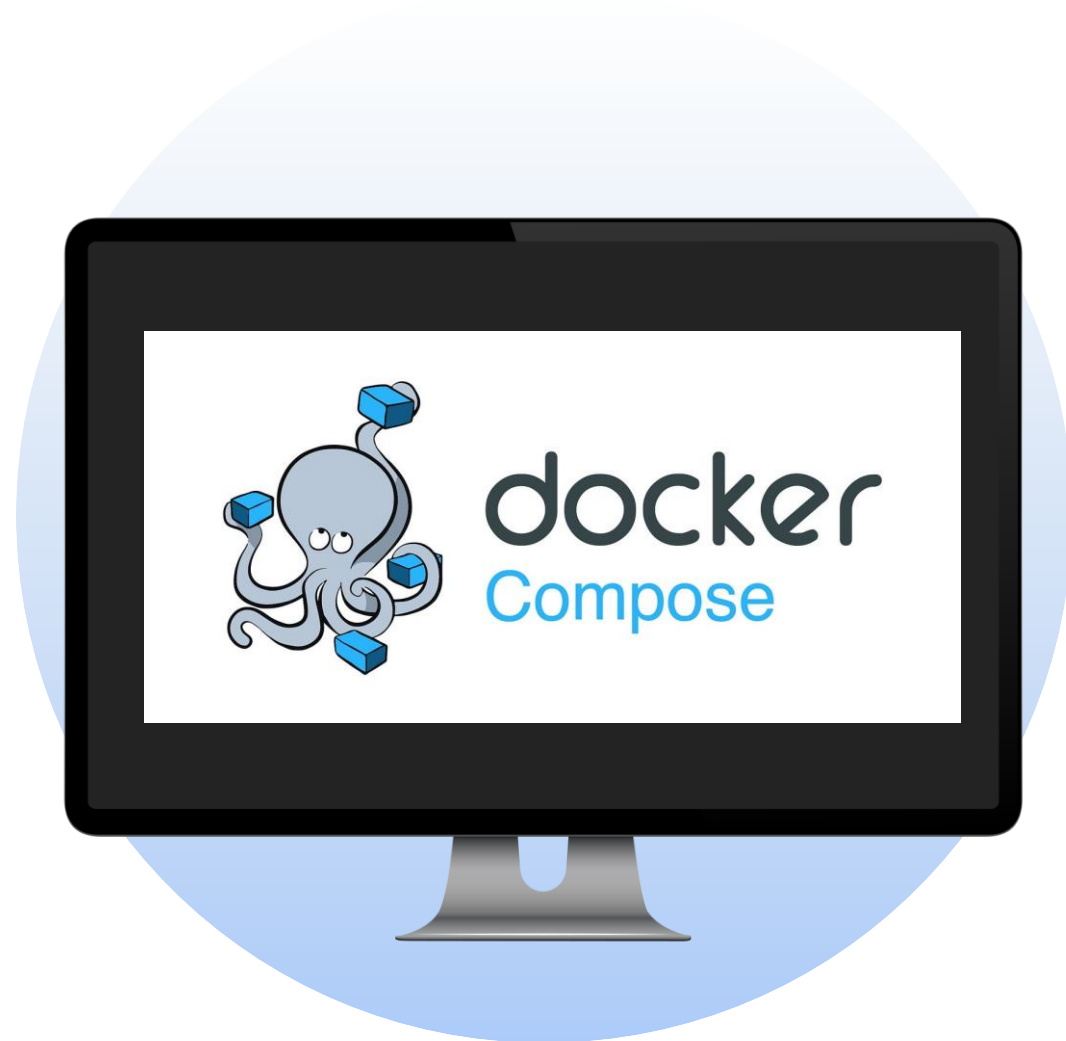
```
yaml
person:
  name: John Doe
  age: 30
  address:
    street: 123 Main St
    city: Anytown
```

Lists of Key-Value Pairs:

You can combine lists with key-value pairs:

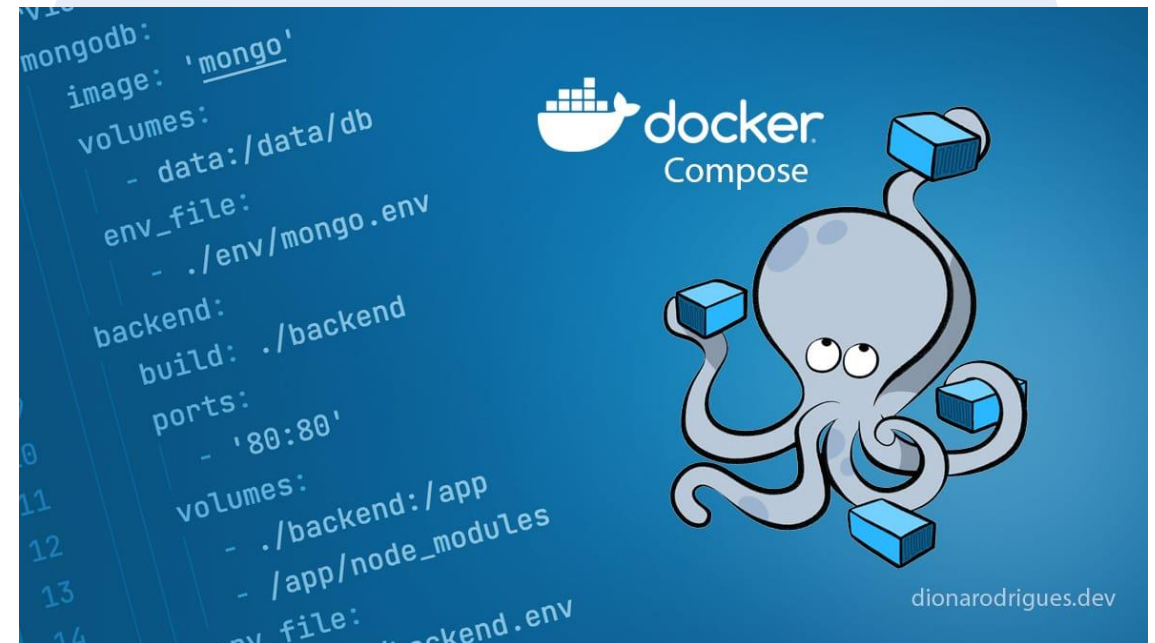
```
yaml
contacts:
  - type: email
    value: john.doe@example.com
  - type: phone
    value: +123456789
```

Docker Compose



Docker compose

- Main functionality of Docker compose tool :
 - Run multi-container apps
 - Map volumes based on file configuration
 - Create a network between the containers
 - Port forward using configuration
 - Define environment variables
 - And more



Docker compose - NGinx

Let's run Nginx based on Docker compose

1. Create new file "nginx-docker-compose"
2. Update the file with the following

```
version: '3'

services:
  nginx:
    image: nginx:latest
    container_name: my_nginx_example_3
    ports:
      - "9005:80" # Maps port 9005 on your host to port 80 in the container
    volumes:
      - /mnt/c/html:/usr/share/nginx/html # Mount local directory to Nginx HTML directory
    environment:
      - KAFKA_SERVER=130.0.1.1 # Environment variable
    restart: always # Ensure the container restarts if stopped
```

3. Run > **docker compose -f nginx-docker-compose up**

Docker compose - NGinx

4. Open the browser <http://localhost:9005>
 - a. This is the port that we have mapped in the docker compose file - you can change it
5. Run > **docker ps** ; verify that the container is running
6. Run > **docker exec -it nginx-docker-compose sh**
7. Run > **printenv**
8. Verify that the kafka environment variable you have set is defined

```
root@IL-DZBEDA:~# docker exec -it nginx-docker-compose sh
# printenv
HOSTNAME=b0dd41c3a9e8
HOME=/root
PKG_RELEASE=1~bookworm
KAFKA_SERVER=130.0.1.1
```


Kubernetes Agenda



Tech Evolution

Kubernetes Basics

Kubernetes Architecture

Kubernetes Practice

Additional Terms

Technology Evolution



51 Once Upon A time

1981 – MS Dos

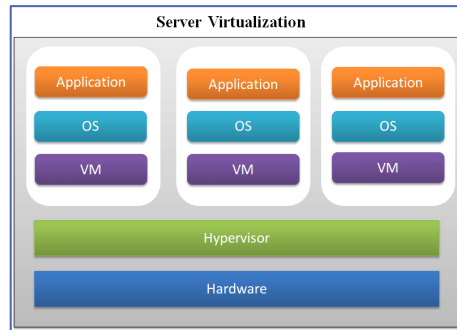
1985 – Windows1

1991 - Unix



1999 – Vmware workstation

2006 – Vmware server



2013 – Docker

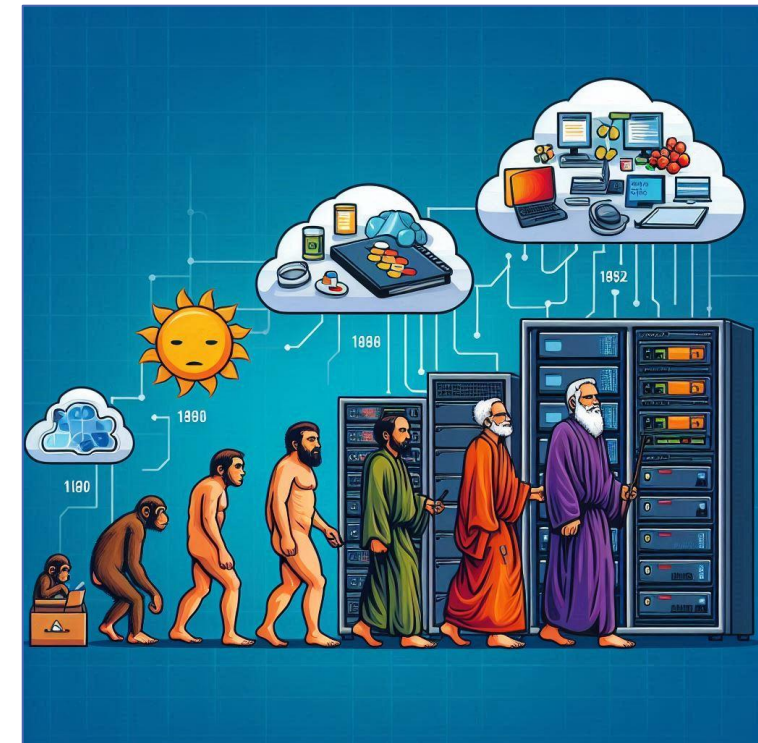
2015 – Docker swarm



2015 - Kubernetes



kubernetes




Kubernetes

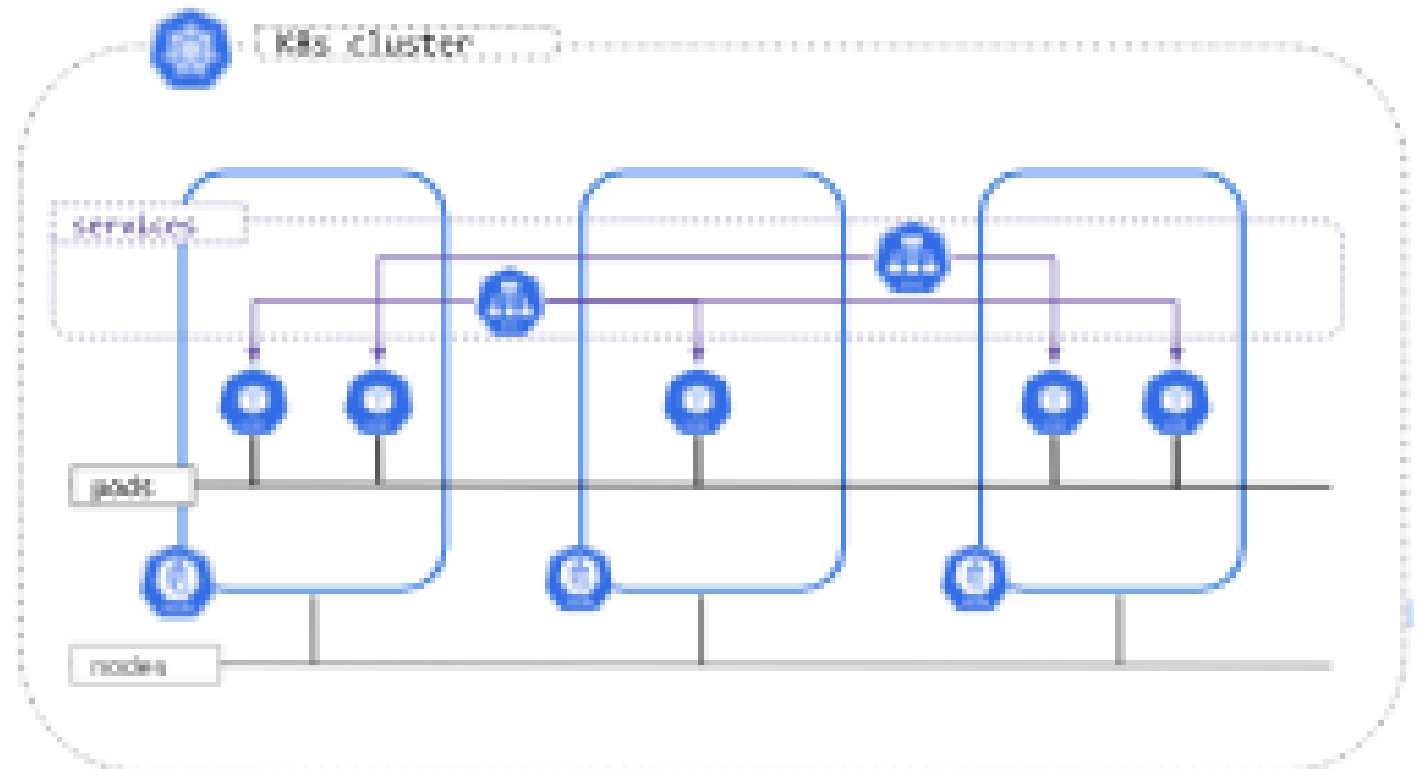
- The What
- The Why
- The How




53 What is Kubernetes?

- Open-Source platform
- Developed by google
- Resource management
 - Automate deployment & scaling
 - Container management

+ 
kubernetes



Why do we need Kubernetes?

- + **Container Orchestration:** Kubernetes automates the deployment, management, scaling, and networking of containerized applications. This is crucial as applications become more complex, with multiple microservices running in containers across various environments
 - + **Scalability:** Kubernetes allows you to easily scale applications up or down based on demand. It automatically adjusts the number of running containers, ensuring optimal resource utilization.
 - + **High Availability and Reliability:** By managing the distribution of containers across multiple nodes (machines), Kubernetes ensures that your application remains available even if some nodes fail. It automatically restarts failed containers and can move workloads to healthy nodes.
 - + **Efficient Resource Utilization:** Kubernetes intelligently schedules containers based on available resources and the requirements of each container, optimizing the use of computing resources like CPU and memory.
- 
- A large, light blue circle is positioned on the right side of the slide. Three light blue plus signs are scattered in the lower right area, with one plus sign located near the bottom center and two others further to the right, partially overlapping the blue circle.

Why do we need Kubernetes?

- **Automated Rollouts and Rollbacks:** Kubernetes enables seamless updates to your application. It can perform rolling updates, ensuring that new versions are deployed gradually, minimizing downtime. If an update fails, Kubernetes can automatically roll back to a previous stable version.
- **Self-Healing:** Kubernetes monitors the health of containers and automatically replaces or restarts them if they fail or become unresponsive. This reduces the need for manual intervention and ensures that your application remains stable.
- **Platform Independence:** Kubernetes is cloud-agnostic, meaning it can run on any cloud provider or on-premises infrastructure. This flexibility allows you to avoid vendor lock-in and run your applications wherever it makes the most sense.

Kubernetes Distribution

- **Open shift** – Redhat
- **Tanzu** – Vmware
- **RKE2** – Rancher
- **Kubernetes Vanila** – Google
- **EKS** – Amazon cloud
- **GKE** – Google **cloud**
- **Mimikube** – Small Based on Docker



57

Kubernetes – High Level Architecture – Hardware Level

Control Plane

Master 1

Master 2

Master 3

Nodes \ Data Plane

Worker 1

Worker 2

Worker 3

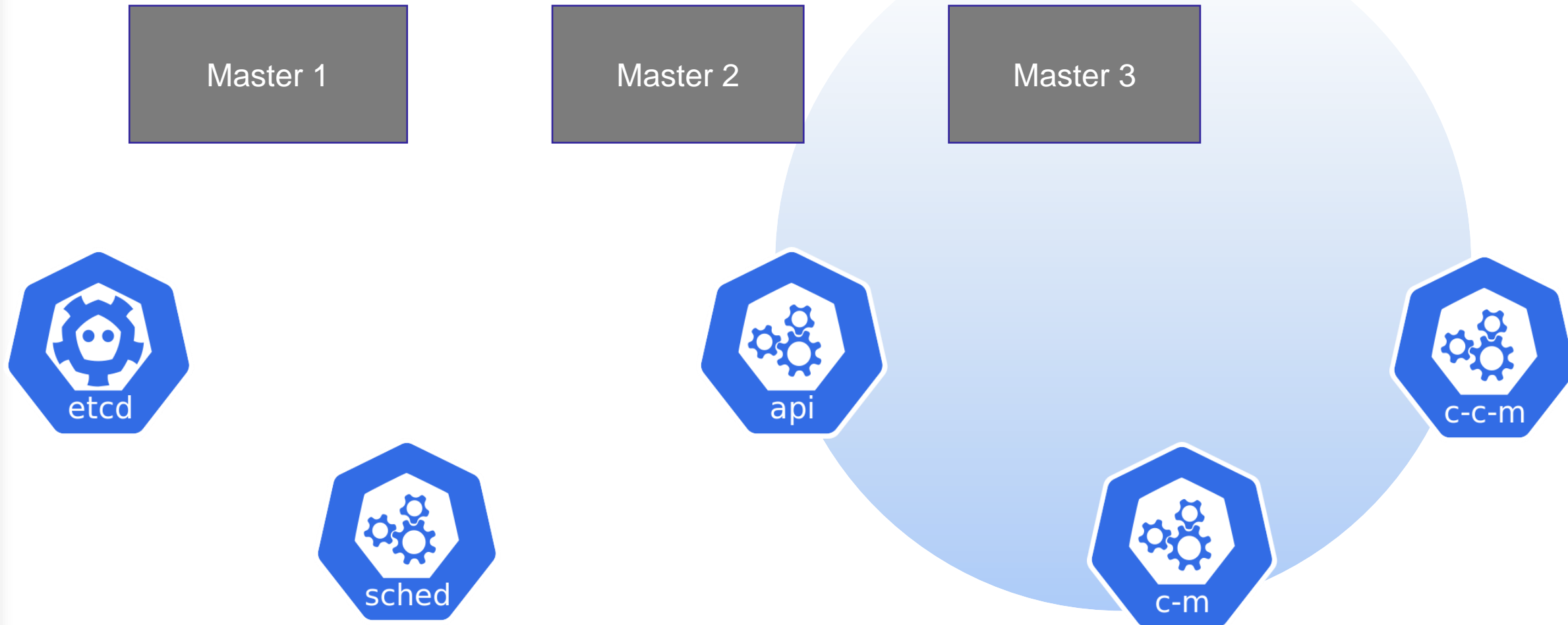
Worker 4

Worker N



Kubernetes – Control plane main components

Control Plane



Kubernetes – Control plane main components

➤ Etcd:

- Kubernetes DB
- key-value store:
- Highly available
- Performance-Critical
- Cluster State Management



➤ Scheduler:

- Workload Placement :The Scheduler is responsible for placing pods on nodes within the cluster
- Resource Awareness: Allocation is based on required resources (CPU , Memory , Storage)
- Affinity and Anti-Affinity : Pod placing based on rule (Labels , presence of other pods)



Kubernetes – Control plane main components

➤ API:

- The API Server acts as the central management hub for the Kubernetes Control Plane, handling all requests to and from the cluster.
- Manages security by handling user authentication and authorization, ensuring only permitted actions are performed on the cluster.(RBAC)



➤ Control Manager:

- Continuously monitors the desired state versus the current state of the cluster and takes corrective actions to ensure the cluster's state matches the desired configuration
- Includes several core controllers such as the **Replication Controller** (manages pod replication), **Node Controller** (monitors node availability) and more.



➤ Cloud Control Manager:

- Requires for cloud integration



61

Kubernetes – Data plane main components

Nodes \ Data Plane

Worker 1

Worker 2

Worker 3

Worker 4

Worker N



Container Runtime



Kubernetes – Data plane main components



➤ Kubelet:

- Run on each worker and master nodes
- ensures that containers are running in pods by communicating with the Kubernetes API server Highly available
- interacts with container runtimes (like Docker or containerd) to manage the lifecycle of containers on the node
- performs health checks (liveness and readiness probes) on containers to ensure they are functioning properly. It restarts containers if they fail these checks
- monitors the resource usage of the node, including CPU, memory, and storage, reporting back to the control plane to help with resource scheduling
- Managing Pod Level Network - helps manage networking for pods using CNI plugins, enabling networking between different pods across nodes



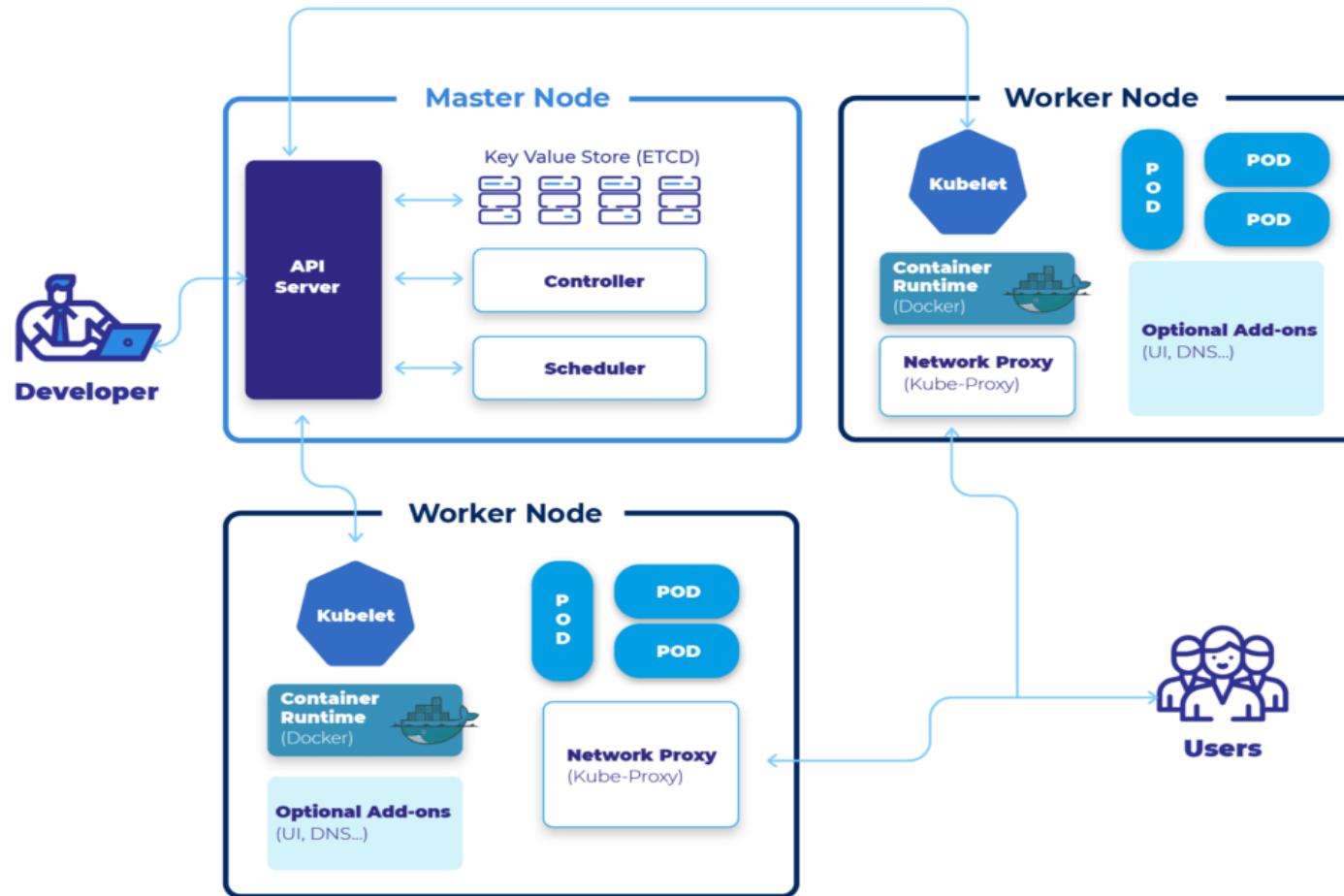
Kubernetes – Data plane main components

➤ Kube-proxy:

- Managing Service level Networking
- responsible for maintaining network rules on each Kubernetes node. It manages the network rules for enabling communication between different services, pods, and external clients
- Handling Network policy

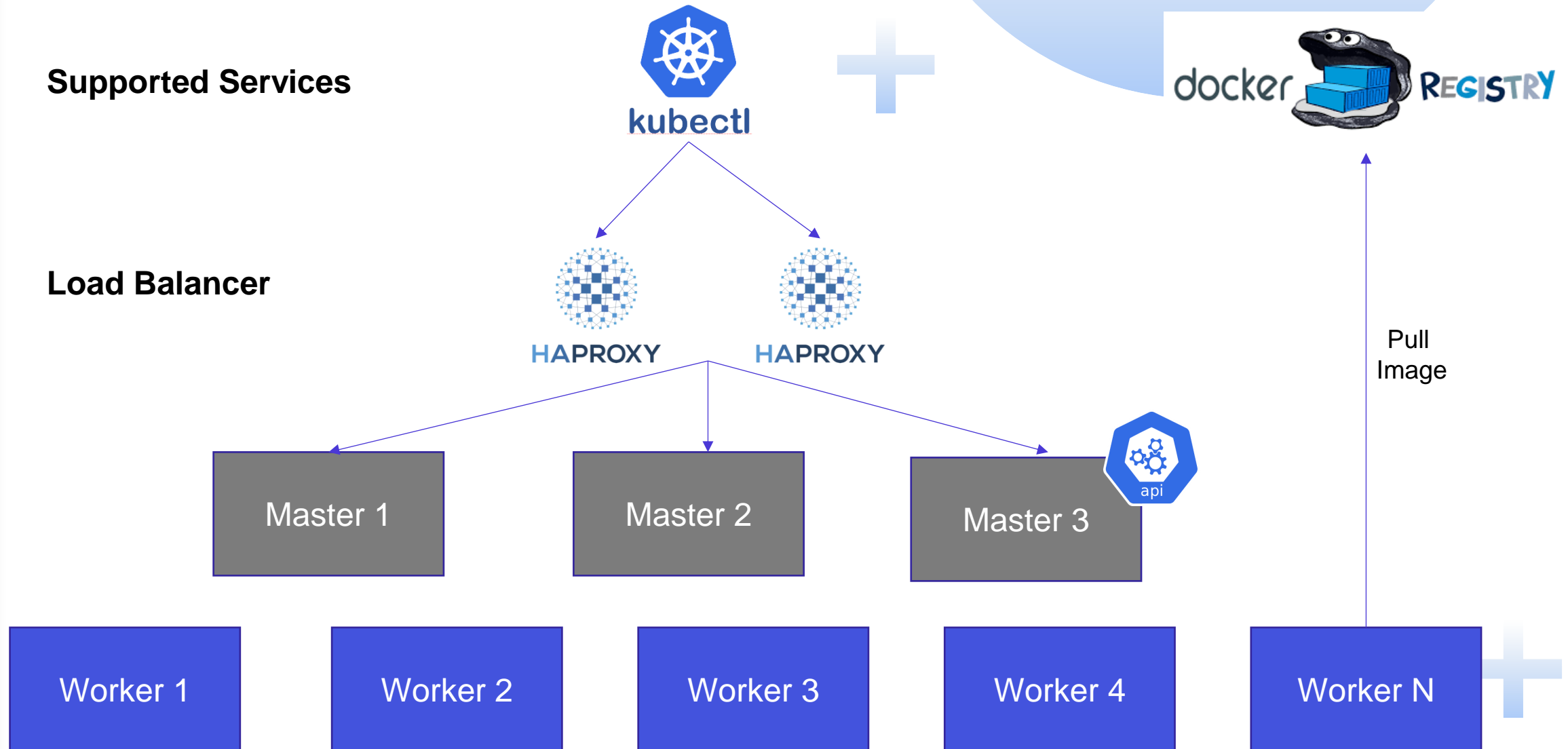


Kubernetes – High Level Architecture – Flow



65

Kubernetes – High Level Architecture – External Services



Kubernetes Practice



Create you first Kubernetes cluster

- Start WSL
- Run > `minikube delete`
- Run > `minikube start --nodes=3 --driver=docker`
- Run > `minikube status`
- Run > `minikube kubectl -- get nodes`



Kubernetes - kubectl

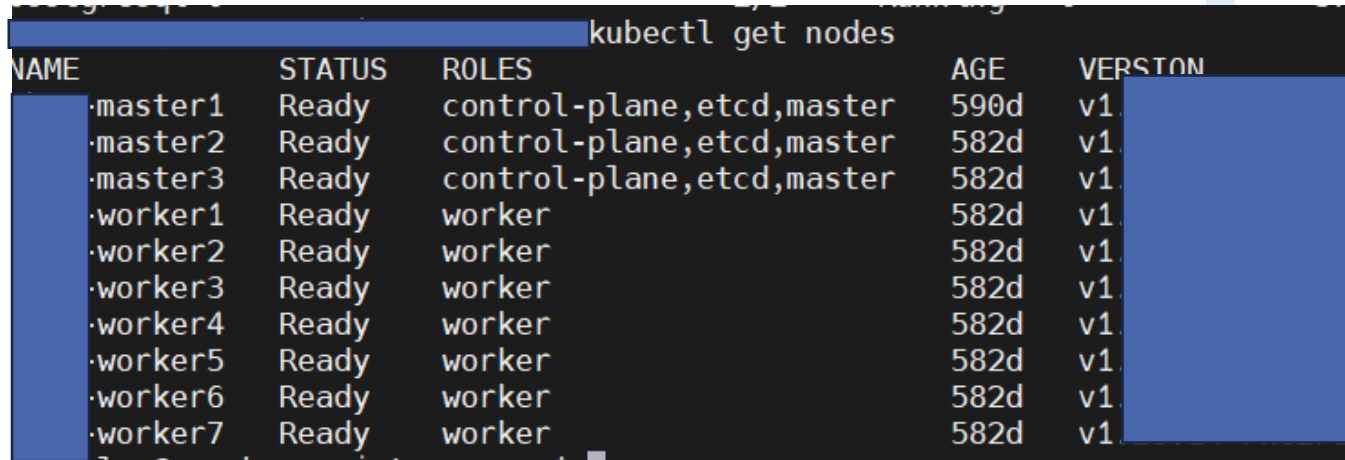
➤ kubectl:

- Kubectl is the command-line tool used to interact with Kubernetes clusters. It allows users to manage Kubernetes resources and applications, perform various administrative tasks, and view the state of the cluster.
- Kubectl interacts with Kubernetes API
- Configuration file is located under /home/<user>/.kube/config
- You can install auto-complete for kubectl – Check the Tips section
- Basic commands (example)
 - Deploy component: **kubectl apply -f <component-file>.yaml**
 - Get resource: **kubectl get <pods\service\deployment\ingress\configmap\secret..>**
 - View logs: **kubectl logs -f <pod-name>**
 - Interact with pod: **kubectl exec -it <pod-name> -- /bin/bash**
 - Troubleshooting pod: **kubectl describe pod <pod-name>**
 - Create configmap: **kubectl create configmap my-config --from-file=<file-name>**



Cluster configuration

- Get nodes in cluster: > **kubectl get nodes**



```
kubectl get nodes
```

NAME	STATUS	ROLES	AGE	VERSION
master1	Ready	control-plane,etcd,master	590d	v1
master2	Ready	control-plane,etcd,master	582d	v1
master3	Ready	control-plane,etcd,master	582d	v1
worker1	Ready	worker	582d	v1
worker2	Ready	worker	582d	v1
worker3	Ready	worker	582d	v1
worker4	Ready	worker	582d	v1
worker5	Ready	worker	582d	v1
worker6	Ready	worker	582d	v1
worker7	Ready	worker	582d	v1

- Check node in cluster: > **kubectl describe node rke-worker7**
- Get cluster events: > **kubectl get events --all-namespaces**
- Check resource per node: > **kubectl get nodes -o custom-columns=NAME:.metadata.name,CAPACITY_CPU:.status.capacity.cpu,CAPACITY_MEMORY:.status.capacity.memory**
- Check running k8s services: > **kubectl get pods -n kube-system**



➤ POD

In Kubernetes, a **Pod** is the smallest and simplest deployable unit. It represents a single instance of a running process in your cluster. A pod can contain one or more containers that **share the same network namespace**, storage, and configuration. **pods** are ephemeral and can be replaced at any time



➤ Deployment

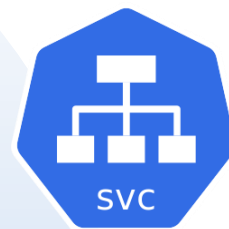
A **Deployment** in Kubernetes is a higher-level abstraction that manages the lifecycle of pods, ensuring that the desired number of pod replicas are running at all times. It provides features for scaling, rolling updates, and rollbacks, making it easier to manage application workloads



Terms

➤ Service

A **Service** in Kubernetes is an abstraction that defines a logical set of pods and a policy for accessing them. Services provide a stable endpoint (IP address or DNS name) to access a group of pods, which is especially useful since pods are ephemeral and can be replaced at any time. Services decouple the consumers of a service from the backend pods, making it easier to manage dynamic workloads.



➤ Namespace

A **Namespace** in Kubernetes is a way to divide cluster resources between multiple users or teams, allowing for more granular control over resources, isolation, and better management of complex environments



YAML - YML

YAML is a versatile, human-readable data serialization language commonly used for writing configuration files.

The presentation of data is based on Key-Value pair

- + To create resource in Kubernetes we will need to apply it using YAMK configuration file

Note: Do not use TAB spacing in YAML !!!




```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  namespace: default
  labels:
    app: nginx
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
        - name: nginx-container
          image: nginx:1.19.2
          ports:
            - containerPort: 80
```

The What ; Deployment , Service , POD , ConfigMap , Secret

The What's metadata ; Names , Labels in Namespace

The What's Spec ; Replicas , Container , Volumes , ENV

Note:
Every What kind will have a different spec

Nginx - Namespace & pods

- Start Minikube with 3 nodes
 - Delete current cluster: **minikube delete**
 - Start minikube cluster : **minikube start - -nodes 3**
- How to create a POD?
 - Deploy: > **kubectl apply -f nginx-pod.yml** – Did you manage to deploy the pod ?
 - Create namespace by running > **kubectl create namespace class1**
 - Get namespace: Run > **kubectl get namespaces**
 - Deploy: > **kubectl apply -f nginx-pod.yml**
 - Get pods: Run > **kubectl get pods -A**
 - Get pods: Run > **kubectl get pods -n class1**
 - Delete pod: Run > **kubectl delete pod -n class1 nginx-pod**
 - Get pods: Run > **kubectl get pods -n class1** ; Can you see the POD ?

Note:

If a namespace is not defined in the POD deployment , the “default” namespace will be used

Nginx - deployment

- Prerequisites - Deploy utility pod by running the following commands
 - Deploy: > **kubectl apply -f utility-pod.yml**
 - Verification: > **kubectl get pods -A -o wide**; verify utility pod is in running mode

- Deploy Nginx Deployment by running the following commands
 - Deploy: > **kubectl apply -f nginx-deployment-simple.yml**
 - Get pod: > **kubectl get pods -A**; verify Nginx pod is in running mode. **On which namespace it is running & why ?**
 - Delete pod: > **kubectl delete pod nginx-deployment-simple-xxxx** ; verify utility pod is in running mode
 - Get pod: > **kubectl get pods -A -o wide**; **Do you see any issue ??**
 - Get deployment: > **kubectl get deployment**
 - Describe deployment : > **kubectl describe deployment nginx-deployment-simple**
 - Get ReplicaSet : > **kubectl get replicaset**
 - Describe ReplicaSet : > **kubectl describe replicaset nginx-deployment-simple-XXX**
 - Delete deployment: > **kubectl delete deployment nginx-deployment-simple**
 - Get pod: > **kubectl get pods -A**; **Do you see any issue ??**
 - Get ReplicaSet : > **kubectl get replicaset**

Nginx - Scale up

- Deploy nginx: > **kubectl apply -f nginx-deployment-simple.yml**
- Get pod: > **kubectl get pods -A**; verify Nginx pod is in running mode
- Get deployment: > **kubectl get deployment**
- Delete pod: > **kubectl edit deployment nginx-deployment-simple**
 - Under spec section update the replicas from 1 to 2 & save like in vi (esc + :wq! + Enter)

```
spec:
  progressDeadlineSeconds: 600
  replicas: 2
  revisionHistoryLimit: 10
  selector:
    matchLabels:
      app: nginx
  strategy:
:wq!
```




- Get pod: > **kubectl get pods -o wide**; verify Nginx pod is in running mode
- Lets take on Node down ; Run > **kubectl drain <node-name> --ignore-daemonsets --delete-emptydir-data**
 - What happed
 - kubectl get nodes
 - kubectl get pods -o wide
 - kubectl describe replicaset nginx-deployment-simple-xxx
 - Bring the node back ; Run > **kubectl uncordon <node-name>**

Nginx - Scale up

- Get pod: > **kubectl get pods -o wide**; verify Nginx pod is running in 2 replicas and check on which nodes
- Delete pod: > **kubectl delete pod nginx-deployment-simple-xxx**
- Get pod: > **kubectl get pods -o wide**; verify Nginx pod is running in 2 replicas and check on which nodes
 - What happened ?
- Delete deployment: > **kubectl delete deployment nginx-deployment-simple**

- Get pod: > **kubectl get pods -o wide**;

Nginx - Service – cluster IP

- 
- 
- Lets connect to the NGINX
 - Deploy: > **kubectl apply -f nginx-deployment-simple.yml**
 - Get pod: > **kubectl get pod** ; grep the nginx-pod-name
 - Port Forward: > **kubectl port-forward <nginx-deployment-simple-xxx> 9000:80 --address 0.0.0.0**
 - Open browser and connect to <http://localhost:9000>
- 

Nginx - Service – cluster IP

How application running in Kubernetes can connect to NGINX **service** ?

- Describe pod: > **kubectl describe pod nginx-deployment-simple-xxxx** What is the IP assigned to the pod?
- Delete pod: > **kubectl delete pod nginx-deployment-simple-xxx**
- Describe pod: > **kubectl describe pod nginx-deployment-simple-xxxx** What is the IP assigned to the pod?
- Get Services: > **kubectl get service -A**
- Deploy Service for supporting nginx pod: > **kubectl apply -f nginx-service-clusterip-simple.yml**
- Get Services: > **kubectl get service -A** ; What do you see ?
- Run > **kubectl exec -it utility-pod -- /bin/bash**
 - Run **curl -v http://<the IP was assigned to “nginx-service-clusterip-simple” service>**
 - Run **curl -v http://nginx-service-clusterip-simple**

Nginx - Service – Nodeport

➤ What is Service from type **NodePort** ?

A Service of type **NodePort** in Kubernetes exposes a service on a static port on **each node's** IP address. This allows external access to the service from outside the Kubernetes cluster. It is useful for scenarios where you want to provide access to your service from outside the cluster without relying on an external load balancer.

This is great solution for lab – Do not use it in production !!!

Nginx - Service – Nodeport

This excises might not be supported by Minikube without special tunneling configuration

How application running in kuberntes can connect to NGINX **service** via nodeport?

- Apply Deployment **kubectl apply -f nginx-deployment-simple.yml**
- Get Deployment **kubectl get deployment**
- Get Deployment **kubectl get deployment -n class1**
- Get Services: > **kubectl get service -A**
- Deploy Nodeport Service for supporting nginx pod: > **kubectl apply -f nginx-service-nodeport-simple.yml**
- Get Services: > **kubectl get service -A**
- Open browser and connect to <http://130.1.1.1:30007/> ; This is the IP of one of the Kubernetes nodes
 - In Nodeport you can use ports range between 30000 – 32767
 - The port will be available on any node that is part of the cluster
 - Kubernetes services are not implemented as processes listening on a specific port, therefore you will not see it using netstat
- Delete resource > **kubectl delete -f nginx-deployment-simple.yml**
kubectl delete -f nginx-service-nodeport-simple.yml

Applying Deployment + Service



Jenkins

Apply
Deployment

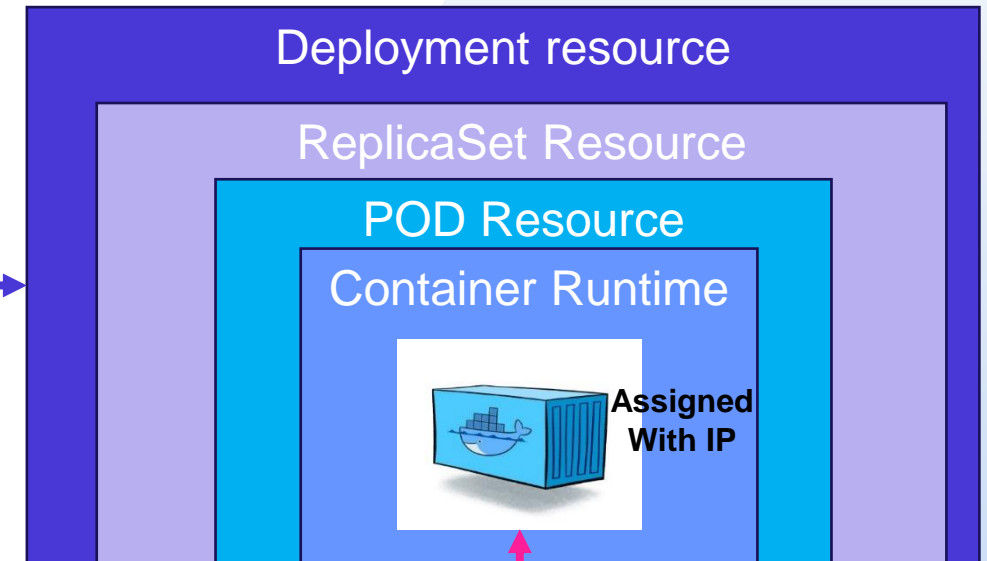
```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment-configmap
spec:
  replicas: 1
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
```

Create
Resource

Apply
Service

```
apiVersion: v1
kind: Service
metadata:
  name: nginx-service-clusterip-simple
spec:
  selector:
    app: nginx
  ports:
    - protocol: TCP
      port: 80
      targetPort: 80
  type: ClusterIP
```

Create
Resource



Service Resource

Type: Cluster IP \ Nodeport

DNS: FQDN Name

Assign to POD

83 Nginx - Simple (Ingress)

➤ What is Ingress?

Ingress is an API object that manages external access to services within a cluster. It acts as a reverse proxy, allowing external HTTP and HTTPS traffic to reach services running inside the cluster without needing to expose those services directly using a NodePort or LoadBalancer

Ingress is the right way to expose pods in production !!!

This excises might not be supported by Minikube without special tunneling configuration

- How application running in Kubernetes can connect to NGINX application via **ingress**?
 - Deploy: > `kubectl apply -f nginx-deployment-simple.yml`
 - Deploy Service: > `kubectl apply -f nginx-service-clusterip-simple.yml`
 - Get Ingress: > `kubectl get ingress`
 - Deploy Ingress: > `kubectl apply -f nginx-ingress.yml`
 - Get Services: > `kubectl get ingress`
 - Display Logs: > `kubectl logs -f nginx-deployment-simple-xxxx`
 - Update /windows/system32/driver/etc/hosts: `130.1.1.1 nginx.class1.com ;` ; This is the IP of one of the Kubernetes nodes
 - Open browser and connect to <http://130.1.1.1> – Did you get something ?
 - Open browser and connect to <http://nginx.class1.com> – And Now ?

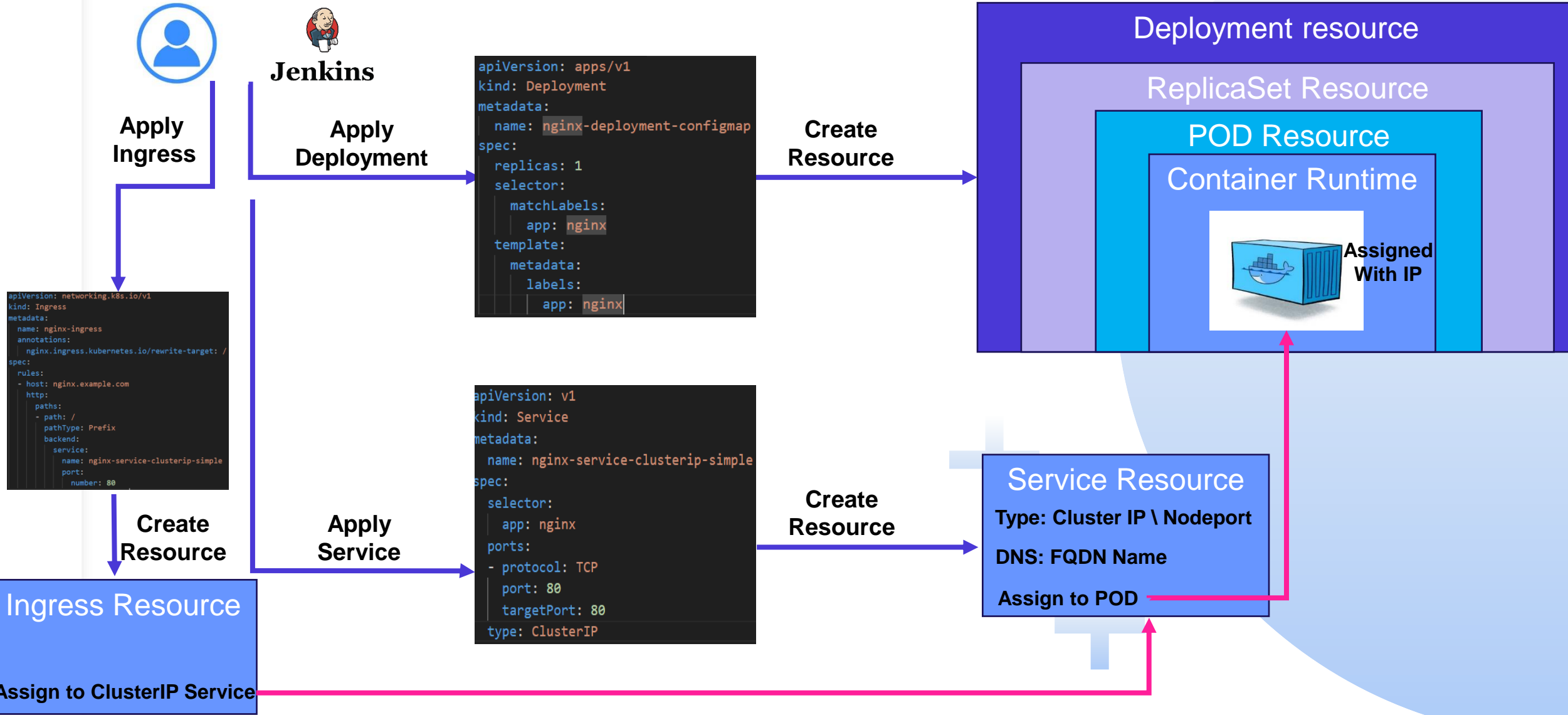
Nginx - Simple (Ingress)

- Scale the Nginx application by update the Deployment > **kubectl edit deployment nginx-deployment-simple**
- Get Pods: > **kubectl get pods ;** – How many pods are running?
- Display Logs on the new pod: > **kubectl logs -f nginx-deployment-simple-xxxx** – How Can we know which is the new POD?
- Open browser and connect to <http://nginx-class1.com>
 - Reconnect for few time and check logs
 - Is the Load balancing works ?
 - Who is responsible for the Load Balancing ?



85

Applying Deployment + Service + Ingress



➤ ConfigMap

In Kubernetes, a **ConfigMap** is a resource used to store configuration data in key-value pairs. It allows you to decouple configuration artifacts from container images, making it easier to manage and update configuration data independently of your applications.

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: my-configmap
data:
  key1: value1
  key2: value2
  This-is-my-file-name: |
    Once Upon a Time .....
```



Nginx - ConfigMap

How to deploy Nginx with ConfigMap and why?

- Deploy : > **kubectl apply -f nginx-deployment-configmap.yml**
- Get pod: > **kubectl get pods -n default**; verify Nginx pod is in running mode. *Is it running?*
- Check pod: > **kubectl describe pod nginx-deployment-configmap-xxxx** ; *Can you see the issue?*
- Deploy Configmap: > **kubectl apply -f nginx-configmap.yml**
- Delete pod: > **kubectl delete pod nginx-deployment-configmap-xxx**
- Get pod: > **kubectl get pods**; verify Nginx pod is in running mode.
- Deploy Service: > **kubectl apply -f nginx-service-clusterip-simple.yml**
- Get Service: > **kubectl get service**
- Connect to the POD: > **kubectl port-forward svc/<service-name> 9000:80 --address 0.0.0.0** ; *Can you see the difference?*
- Open browser and connect to localhost:9000

Nginx - ConfigMap

How to deploy Nginx with ConfigMap and why?

- Get configmap: > **kubectl get configmap**
- Update configmap: > **kubectl edit configmap nginx-configmap**
- Open browser and connect to localhost:9000
 - Is the behavior different than Docker ?
 - Delete pod: > **kubectl delete pod nginx-deployment-configmap-xxx**
 - Reload solution
- Login to Container: > **kubectl exec -it nginx-deployment-configmap-xxx -- sh**
 - Run: **printenv** ; Can you see the environment variable that we have set in the configmap?





➤ Secret

A Kubernetes **Secret** is an object that stores sensitive information, such as passwords, OAuth tokens, SSH keys, and API keys, in a way that is more secure than directly embedding them in configuration files or containers. Secrets allow you to manage sensitive data separately from application code. By default the secret is generated using base64 encoding

```
apiVersion: v1
kind: Secret
metadata:
  name: nginx-secret
  type: Opaque
data:
  password: bXlwYXNzd29yZA== # base64-encoded password
```

➤ How to create a secret using base64 encoding

- Open Linux terminal , like you WSL
- Run the following command `echo -n "your_value" | base64`

```
emperor@IL-DZBEDA:/mnt/c/Users/dzbeda/Desktop/Traning/k8s$ echo -n "mypassword" | base64
bXlwYXNzd29yZA==
```

Different Types of Secrets

```
# Opaque Secret: Default type, stores generic key-value pairs
apiVersion: v1
kind: Secret
metadata:
  name: opaque-secret
type: Opaque
data:
  username: YWRtaW4= # 'admin' base64 encoded
  password: cGFzc3dvcmQ= # 'password' base64 encoded


---

# TLS Secret: Stores TLS certificate and key
apiVersion: v1
kind: Secret
metadata:
  name: tls-secret
type: kubernetes.io/tls
data:
  tls.crt: LS0tLS1CRUdJTiBDRVJUSUZJQ0FUR50tLS0t... # base64 encoded certificate
  tls.key: LS0tLS1CRUdJTiB5SU0EgUFJJVWkFUR5BLRVktLS0t... # base64 encoded private key
```

```
# SSH Auth Secret: Stores an SSH private key
apiVersion: v1
kind: Secret
metadata:
  name: ssh-auth-secret
type: kubernetes.io/ssh-auth
data:
  ssh-privatekey: LS0tLS1CRUdJTiB5SU0EgUFJJVWkFUR5BLRVktLS0t... # base64 encoded SSH private key

---

# Service Account Token Secret: Used for service accounts (auto-generated)
apiVersion: v1
kind: Secret
metadata:
  name: service-account-token-secret
  annotations:
    kubernetes.io/service-account.name: default # Link to the service account
type: kubernetes.io/service-account-token
```



```
# Docker Registry Secret: Used for Docker registry authentication
apiVersion: v1
kind: Secret
metadata:
  name: docker-registry-secret
type: kubernetes.io/dockerconfigjson
data:
  .dockerconfigjson: ewogICJhdXRocyI6IHsKICAgICJodHRwczovL3... # base64 encoded Docker config

---

# Basic Auth Secret: Stores basic authentication credentials (username and password)
apiVersion: v1
kind: Secret
metadata:
  name: basic-auth-secret
type: kubernetes.io/basic-auth
data:
  username: YWRtaW4= # 'admin' base64 encoded
  password: cGFzc3dvcmQ= # 'password' base64 encoded
```



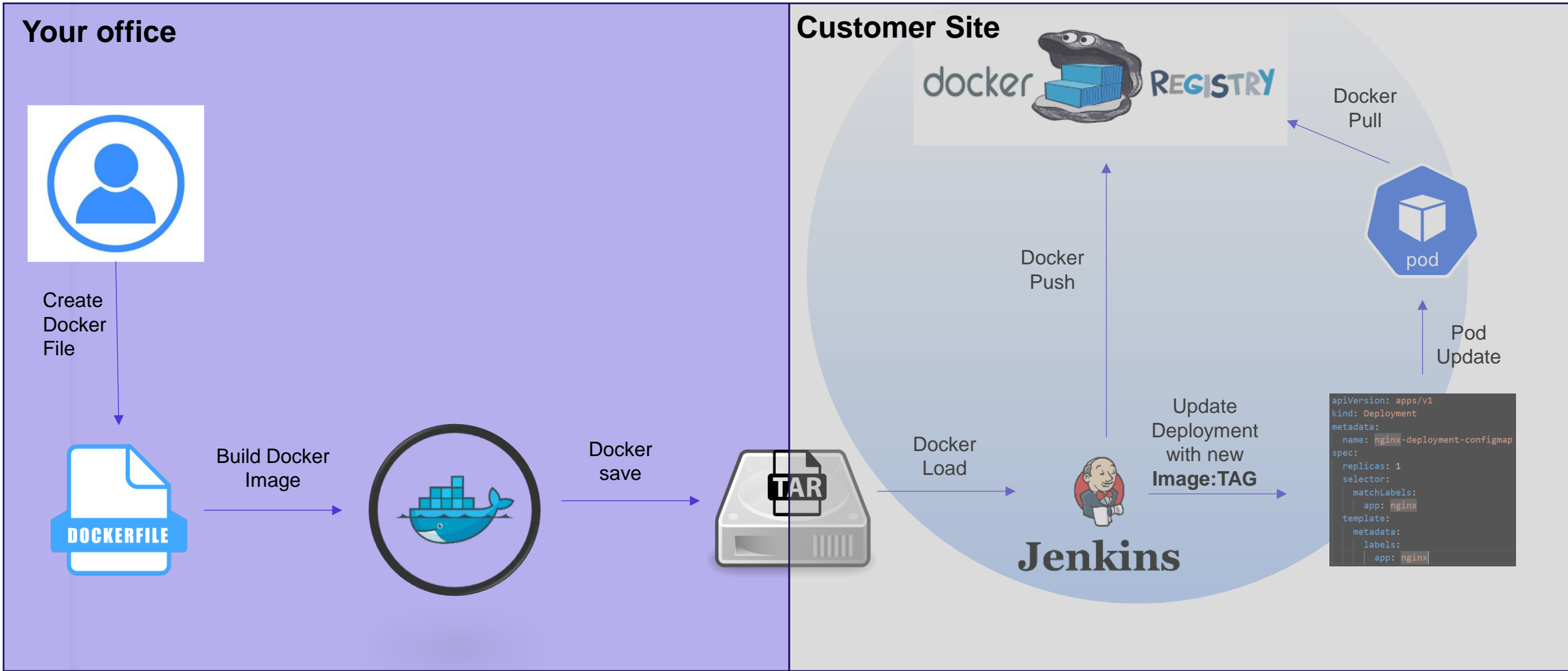
Nginx - Secret

- How to deploy Nginx with Secret and why?
 - Deploy Secret: > **kubectl apply -f nginx-secret.yml**
 - Deploy : > **kubectl apply -f nginx-deployment-secret.yml**
 - Get pod: > **kubectl get pods -n default**; verify Nginx pod is in running mode. Is it running ?
 - Get secret: > **kubectl get secret**
 - Get secret in YAML format: > **kubectl get secrets nginx-secret -o yaml**
 - Login to Container: > **kubectl exec -it nginx-deployment-secret-xxx -- sh**
 - Run: **printenv** ; Can you see the environment variable that we have set in the configmap?



92 Nginx - Application Upgrade

- What dose it mean to upgrade an application that runs in Kubernetes



We don't have time – But worth to mention

➤ **StatefulSet**

A StatefulSet is a Kubernetes workload API object designed to manage the deployment and scaling of a group of Pods where persistent identity and stable storage are critical.

➤ **Upgrade and Rollback**

Upgrade in Kubernetes refers to updating a deployed application to a new version, often using tools like Helm or kubectl to apply changes. Rollback is the process of reverting an application to a previous stable version in case of issues, ensuring minimal downtime and stability.

➤ **Kubernetes Main plugin**

- **CNI(Container Network Interface):** As you mentioned, this is focused on networking and provides a way to configure network interfaces for containers.
- **CSI (Container Storage Interface):** A standard for exposing storage systems to containerized workloads. CSI allows Kubernetes to use various storage solutions seamlessly.

We don't have time – But worth to mention

➤ **PV & PVC**

A Persistent Volume (PV) is a storage resource in Kubernetes that has been provisioned by an administrator or dynamically provisioned using a Storage Class. A Persistent Volume Claim (PVC) is a request for storage by a user, allowing them to consume the storage defined by a PV.

➤ **Helm Chart**

A Helm Chart is a package that contains all the necessary files to deploy an application or service on Kubernetes, including templates, configuration values, and metadata. It simplifies the deployment and management of complex Kubernetes applications by providing reusable and versioned configurations.

➤ **Kubernetes CRD**

Kubernetes Custom Resource Definitions (CRDs) are a way to extend Kubernetes by allowing you to define your own resource types. Think of CRDs as a way to create new objects in Kubernetes that behave like built-in objects (like pods, services, etc.)



Tips

- Install kubectl auto complete
 - **sudo apt-get install bash-completion**
 - **sudo vi ~/.bashrc**
 - Add the following line to the end of the file :
source <(kubectl completion bash)
echo "source <(kubectl completion bash)" >> ~/.bashrc
 - **source ~/.bashrc**
- Can not resolve external DNS after minikueb installation
 - Sudo vi /etc/wsl.conf
 - [network]
generateResolvConf = false
 - Cp /etc/resolv.conf /etc/resolv.conf.orig
 - Vi /etc/resolv.conf
 - nameserver 8.8.8.8
 - Restart WSL

Tips

- Minikube is using his own docker service. If you wish to create and push local images run the following command – Run it after starting minikube
 - **eval \$(minikube docker-env)**
- To return working with the original docker service run the command - **eval \$(minikube docker-env --unset)**



Commands

➤ **Kubectl apply -f <object-file-name>**

Using this command, you can deploy Kubernetes object from different type that are based on yml file

➤ **Kubectl get pods -A**

Using this command. list the pods from **all name-spaces**

➤ **Kubectl get pods -A -o wide**

Using this command. list the pods from **all name-spaces** and show to which Node the POD was assign

➤ **Kubectl get pods -n <name-space name>**

Using this command, list the pods from **specific name-space**

➤ **Kubectl create namespace <name-space name>**

Using this command, will create a new namespace

➤ **Kubectl get namespaces**

➤ Using this command, list all namespaces available in the cluster

Commands

➤ **kubectl port-forward <pod-name> <local-port>:<target-port> --address 0.0.0.0**

Using this command, you can deploy Kubernetes object from different type that are based on yml file

➤ **kubectl get <resource> <resource-name> -o yaml**

Using this command. You can get the resource configuration output in YAML format

Thank you

David (Dudu) Zbeda

[linkedin.com/in/davidzbeda](https://www.linkedin.com/in/davidzbeda)

https://medium.com/@dudu.zbeda_13698