




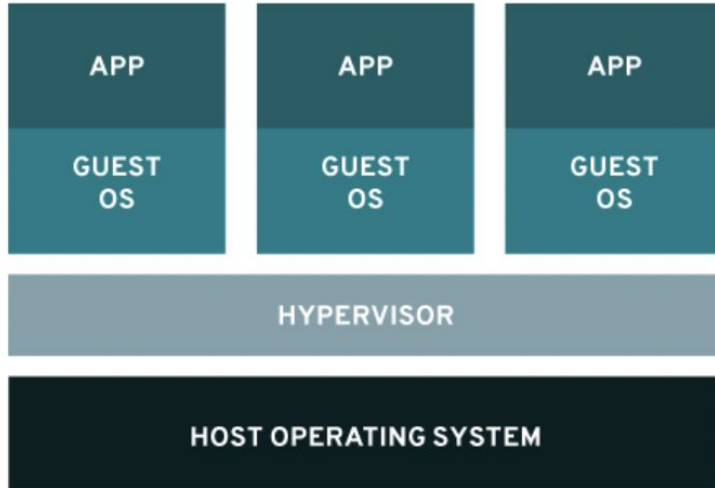
# **Docker & Kubernetes**

Darko Bozhinoski,  
Ph.D. in Computer Science  
Email: [D.Bozhinoski@innopolis.ru](mailto:D.Bozhinoski@innopolis.ru)



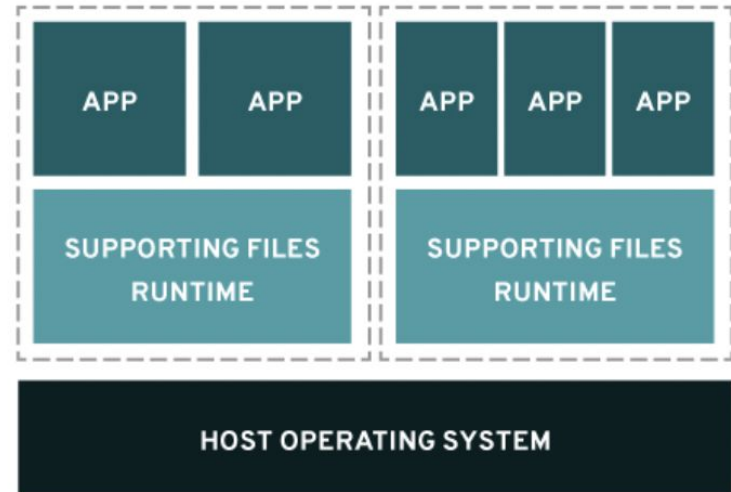
# Containers vs. VMs?

## VIRTUALIZATION



VS.

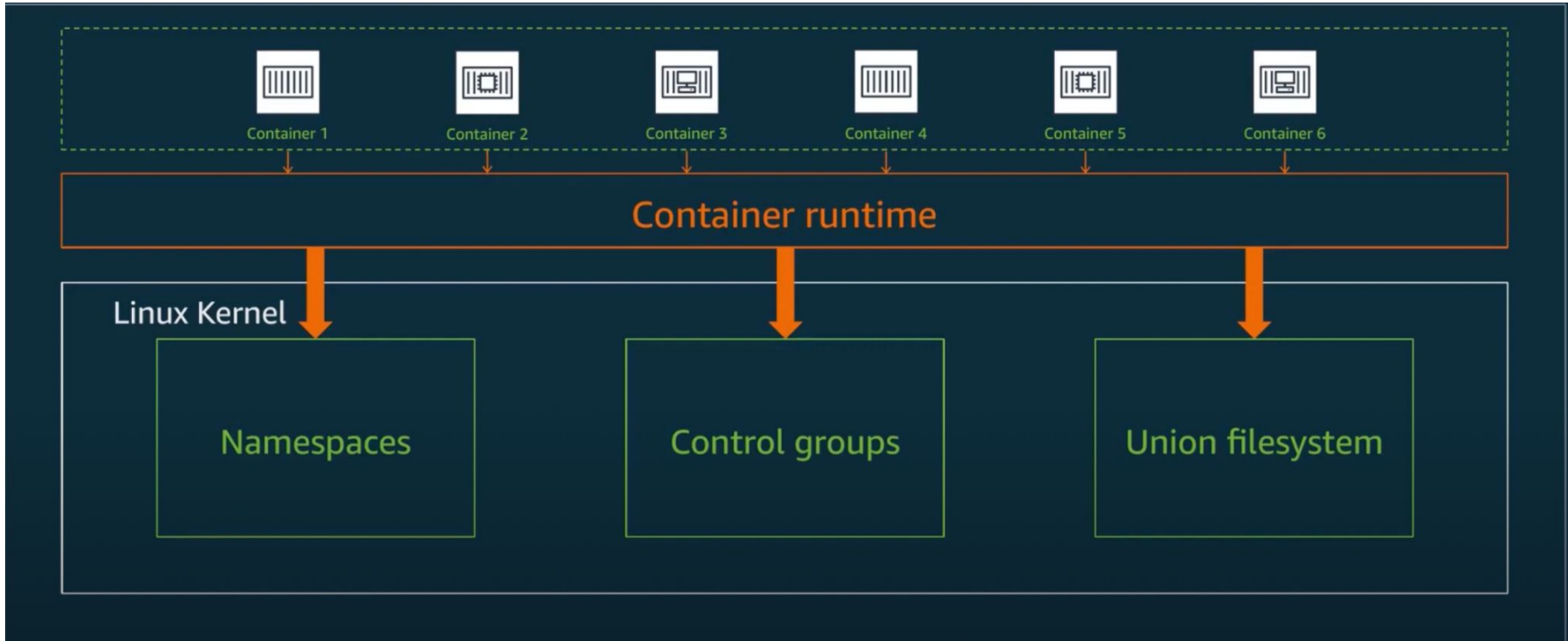
## CONTAINERS



# Linux Containers (LXC)

- "Linux Containers" is a feature to contain a group of processes in an independent execution environment.
- Linux containers are realized with integrating many existing Linux features. There are multiple container management tools such as lxctools, libvirt and docker. They may use different parts of these features.

# LXC Primitives



## Previous lab: Create a container

Creating a process, that is isolated, has its own namespaces, has its own root directory in file system, networking, can save its filesystem as file, and can even limit its own resources.

# Docker

Parameter	LXC	Docker
Data Retrieval	LXC does not support data retrieval after it is processed.	Data retrieval is supported in Docker.
Usability	It is a multi-purpose solution for virtualization.	It is single purpose solution.
Platform	LXC is supported only on Linux platform.	Docker is platform dependent.
Virtualization	LXC provides us full system virtualization.	Docker provides application virtualization.
Cloud support	There is no need for cloud storage as Linux provides each feature.	The need of cloud storage is required for a sizeable ecosystem.
Popularity	Due to some constraints LXC is not much popular among the developers.	Docker is popular due to containers and it took containers to a next level.
Speed Of Deployment	LXC is not lightweight and consumes a lot of time .	Docker Containers are lightweight and fast.

# LXC vs. Docker

In terms of host utilization, Docker differs from LXC in two additional ways:

- It offers an abstraction for machine-specific settings, such as networking, storage, logging, etc. These are part of the
- Docker Engine and make Docker containers are more portable, as they rely less on the underlying physical machine.  
Docker containers are made to run a single process per container.

**Simplicity:** Docker containers were designed specifically for microservices application

**Scalability:** LXC is less scalable compared to Docker.

**Reference:**

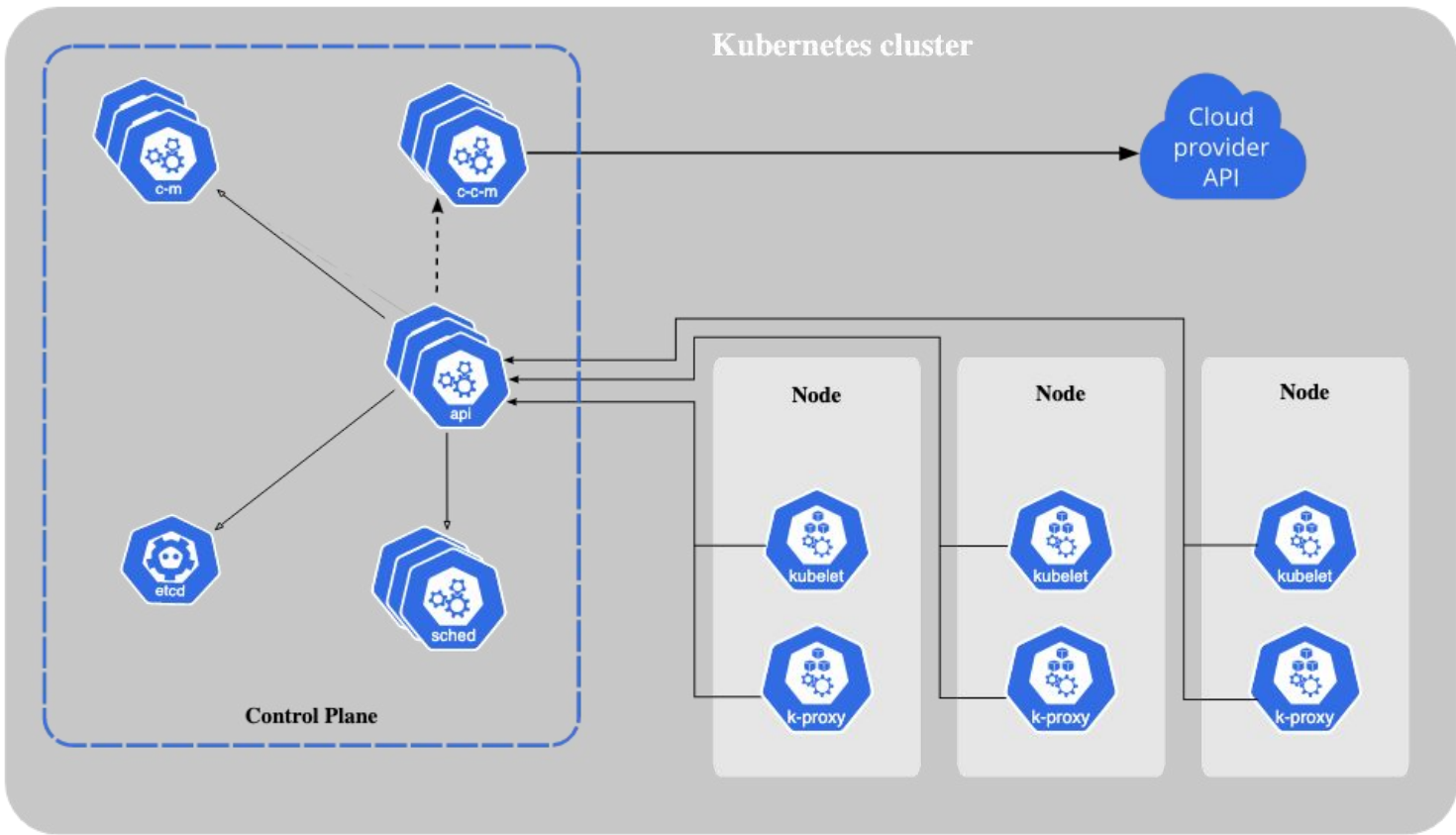
<https://www.upguard.com/blog/docker-vs-lxc#:~:text=Unlike%20LXC%2C%20which%20launches%20an,specified%20by%20a%20docker%20image>



# Security: LXC vs. Docker

LXC is enriched with security configurations such as group policies and a default AppArmor profile to protect the host from accidentally misusing privileges inside the container.

Docker separates the operating system from the services running on it to ensure secure workloads, but the fact that Docker runs as root can increase exposure to malware. This is because the Docker daemon—which manages Docker objects like networks, containers, images, and volumes, and attends to Docker API requests—also runs as root on the host machine. Developers commonly audit Docker installations to identify potential vulnerabilities.



- API server** 
- Cloud controller manager (optional)** 
- Controller manager** 
- etcd (persistence store)** 
- kubelet** 
- kube-proxy** 
- Scheduler** 
- Control plane** 
- Node** 

# Kubernetes

The Control Plane is responsible for managing the cluster. The Control Plane coordinates all activities in your cluster, such as scheduling applications, maintaining applications' desired state, scaling applications, and rolling out new updates.

A node is a VM or a physical computer that serves as a worker machine in a Kubernetes cluster. Each node has a Kubelet, which is an agent for managing the node and communicating with the Kubernetes control plane.

When you deploy applications on Kubernetes, you tell the control plane to start the application containers. The control plane schedules the containers to run on the cluster's nodes. The nodes communicate with the control plane using the Kubernetes API, which the control plane exposes. End users can also use the Kubernetes API directly to interact with the cluster.

Minikube is a lightweight Kubernetes implementation that creates a VM on your local machine and deploys a simple cluster containing only one node. The Minikube CLI provides basic bootstrapping operations for working with your cluster, including start, stop, status, and delete.

<https://minikube.sigs.k8s.io/docs/start/>

# Task 1. Dockerfile

1. Create a dockerfile. (Follow the best practises).
2. Build it and run it.
3. Benchmark your Docker container vs your LXC you created during the last lab
  - use sysbench
  - a. Attributes: CPU total time, FileIO read, FileIO write, Memory access, Thread execution (You can reuse the same LXC runs from the previous assignment).
4. Compare your experience with Docker and LXC. Which one is “better”? Explain why.

Reference: [https://docs.docker.com/develop/develop-images/dockerfile\\_best-practices/](https://docs.docker.com/develop/develop-images/dockerfile_best-practices/)

## Task 2. Dockerize an application

1. Write a rabbitmq reading application (<https://www.rabbitmq.com/>)
2. Create dockerfile
3. Pass variables to configure app (host,port,user,password)
4. Tag and push your image into docker-hub

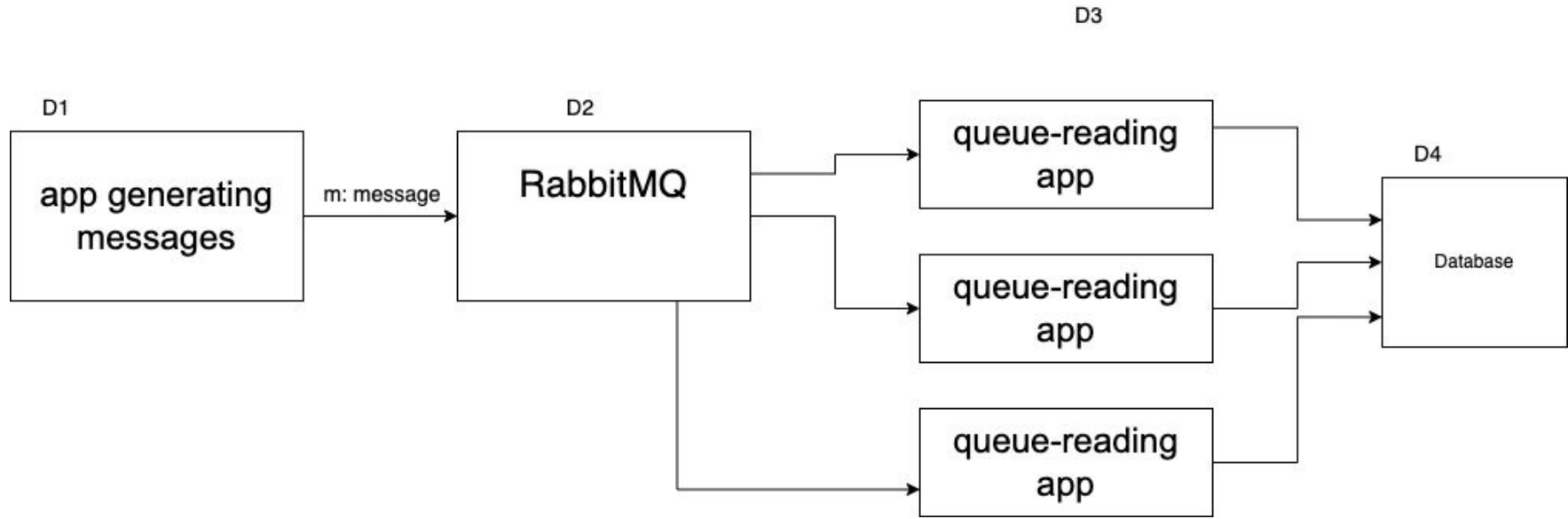
## Task 3. Docker compose

1. Install docker-compose if not installed
2. Create YaML to compose your previously dockerized app and your OWN rabbitmq so they could see each other.
3. Set resource limits for your choice

# Task 4. Docker Orchestration

1. Get k8s: <https://sureshkv1.gitbooks.io/kubernetes-beginners-tutorial/content/basic-exercises.html>  
<https://minikube.sigs.k8s.io/docs/start/>
2. Create following yaml specifications:
  - a. Rabbitmq
  - b. your queue-reading app (you can write your own app or edit one that you have)
  - c. one more app (whitten by you) which will constantly put messages into queue
  - d. postgres database
3. Scenario:
  - a. message - writing app generates messages of 2 types and puts those into queue
  - b. postgres has one db with 2 tables (you decide which ones) that you will fill with data
  - c. rabbitmq obviously stores messages
  - d. depending on message type, your message-reading app will populate one of 2 tables in your postgres database (any 2 tables, you just decide which ones and how to get data for insert from messages)

# Software Architecture of the different services





# Expected Deliverables

1. Write report for all work related to this lab
2. Include description of:
  - a. features you have implemented
  - b. workflow explanation (include all commands you used to make it work)
  - c. Screenshots , if needed