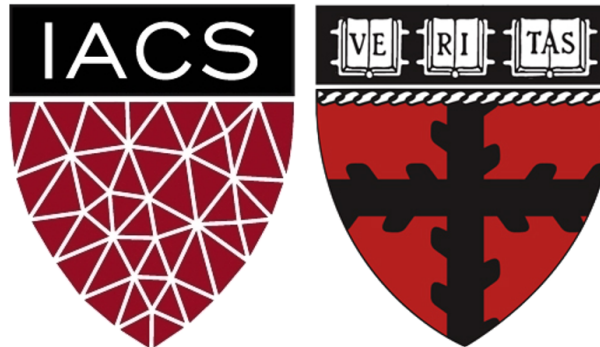


# Lecture 3: Kubernetes

**AC295**

Advanced Practical Data Science

Pavlos Protopapas



# Outline

---

**1: Communications**

2: Recap

3: Introduction to Kubernetes

4: Creating and Running Containers | Review

5: Anatomy of a Kubernetes Cluster

6: Deploying a Kubernetes Cluster

7: Common kubectl Commands

# Communications

---

**Exercise week 1 due today**

**Exercise week 2 will be released today**

# Outline

---

1: Communications

**2: Recap**

3: Introduction to Kubernetes

4: Creating and Running Containers | Review

5: Anatomy of a Kubernetes Cluster

6: Deploying a Kubernetes Cluster

7: Common kubectl Commands

# Recap

## Virtual Environment

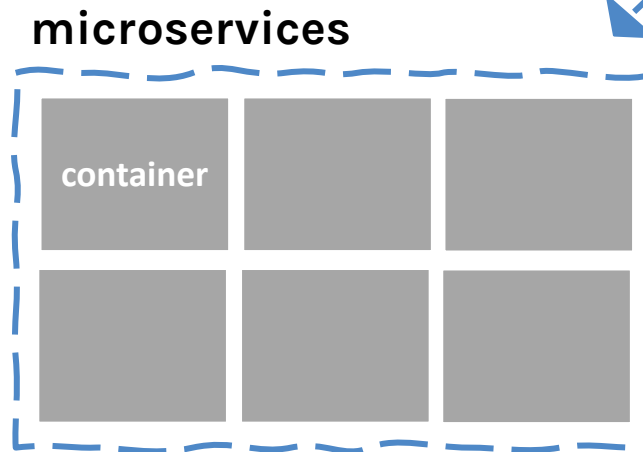
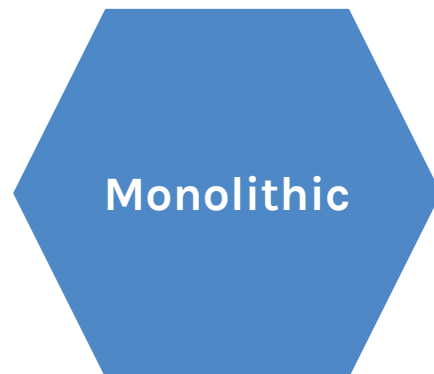
**Pros:** remove complexity  
**Cons:** does not isolate from OS

## Virtual Machines

**Pros:** isolate OS guest from host  
**Cons:** intensive use hardware

## Containers

**Pros:** lightweight  
**Cons:** issues with security, scalability, and control



**How to manage  
microservices?**

# Recap

	VIRTUAL ENV	DOCKER	VM	JH
COMPUTATIONAL COST MEMORY FOOTPRINT	LOW	MEDIUM LOW	HIGH	?
DEPLOYMENT	EASY	MEDIUM	INIT HIGH THEN EASY	N/A
VERSATILITY (TYPES OF APPS)	MEDIUM	MEDIUM HIGH	MEDIUM HIGH	LOW
PORTABILITY	MEDIUM	HIGH	HIGH	HIGH

- COMPUTATIONAL SCIENCE
- DEV OPS
- DATA SCIENCE (NO PIPELINES)
- DATA SCIENCE (PIPELINES)

# Recap

---

We talked about pros/cons of **environments** (removed complexity/does not isolate from OS), **virtual machines** (isolate OS guest from host/intensive use of the hardware), and **containers** (lightweight/issue with security, scalability, and control)

Goal: **find effective ways to deploy our apps** (more difficult than we might initially imagine) and to **break down a complex application** into smaller ones (*i.e. microservices*)

## Issues we have fixed so far:

- conflicting/different operating system
- different dependencies
- "inexplicable" strange behavior

# Outline

---

1: Communications

2: Recap

**3: Introduction to Kubernetes**

4: Creating and Running Containers | Review

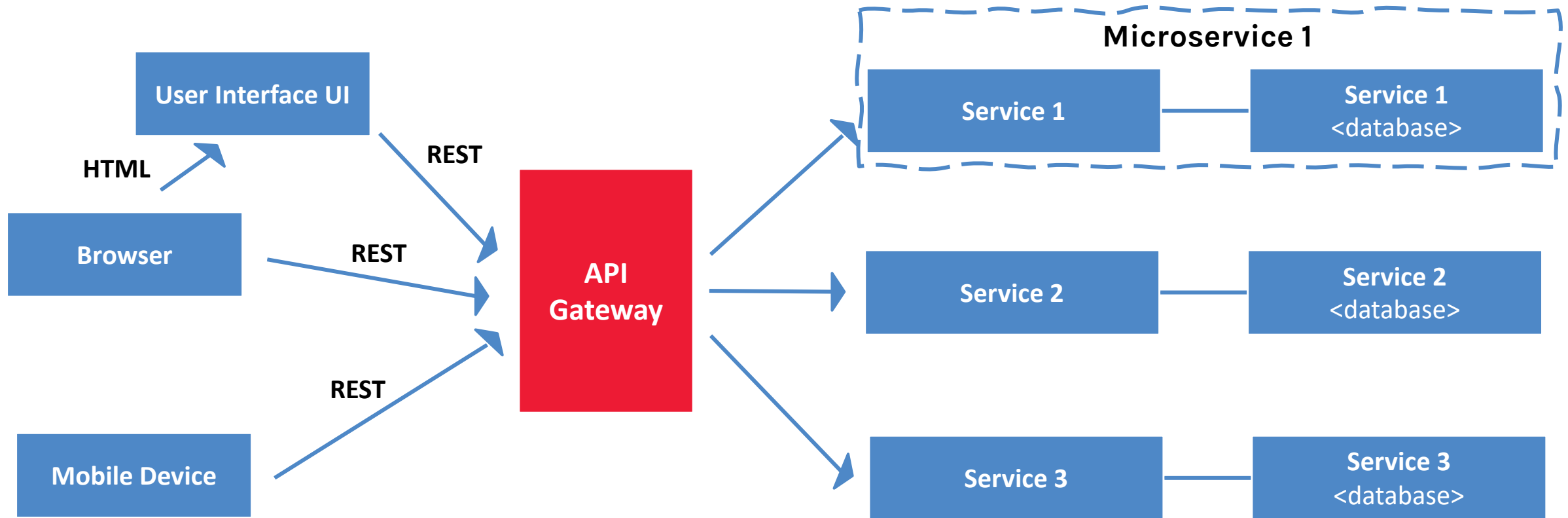
5: Anatomy of a Kubernetes Cluster

6: Deploying a Kubernetes Cluster

7: Common kubectl Commands



# Use Microservice Architecture to build App



TODO > define better application to tell the story:

Service 1 does: a), b), c)

Service 2 does: a), b), c)

Service 2 does: a), b), c)

# Kubernetes to the Rescue <K8s>

---

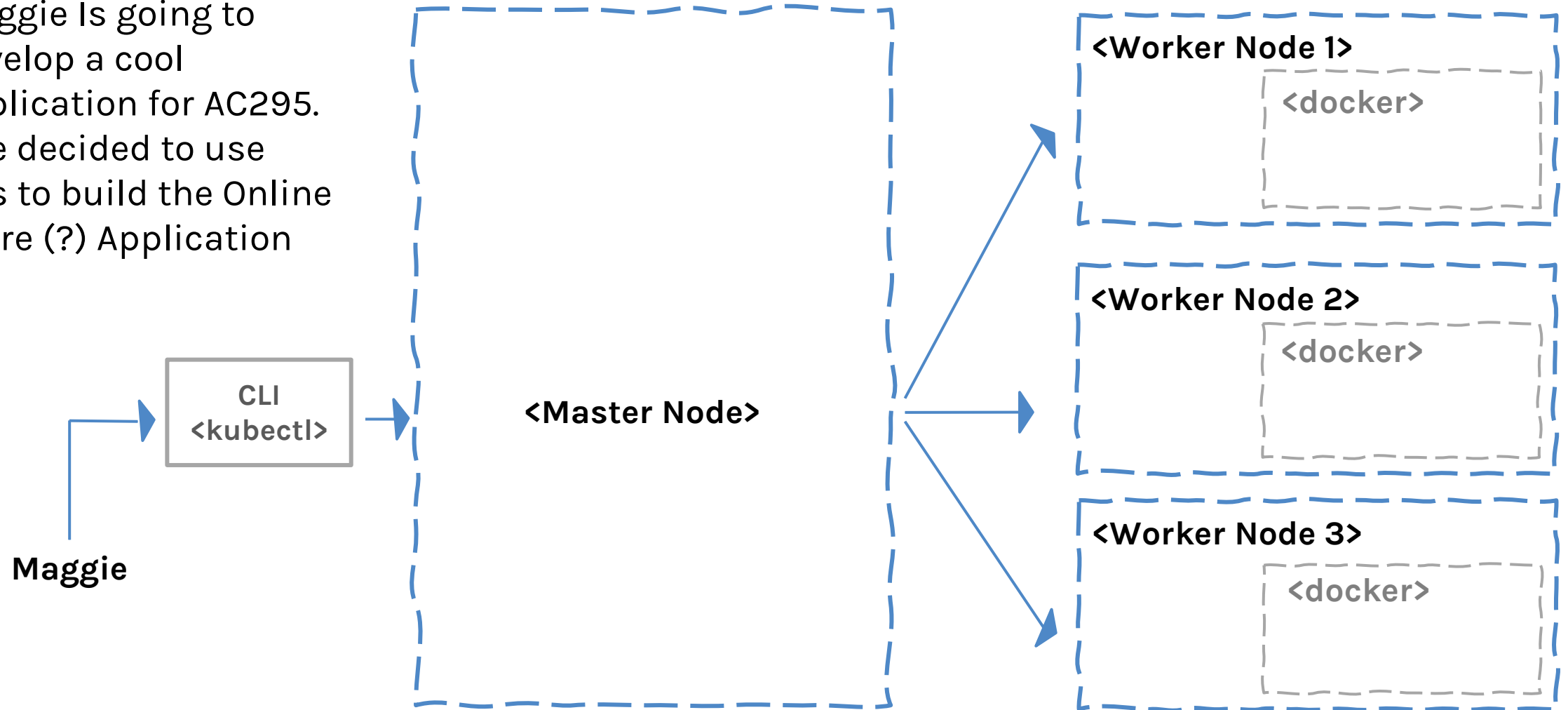


- K8s is an orchestration tool for **managing distributed services** or containerized applications across a distributed cluster of nodes.
- K8s itself follows a **client-server architecture with a master and worker nodes**. Core concepts in Kubernetes include pods, services (logical pods with a stable IP address) and deployments (a definition of the desired state for a pod or replica set).
- K8s **users define rules** for how container management should occur, and then K8s handles the rest!

# How do we build it with K8s? Components & Architecture

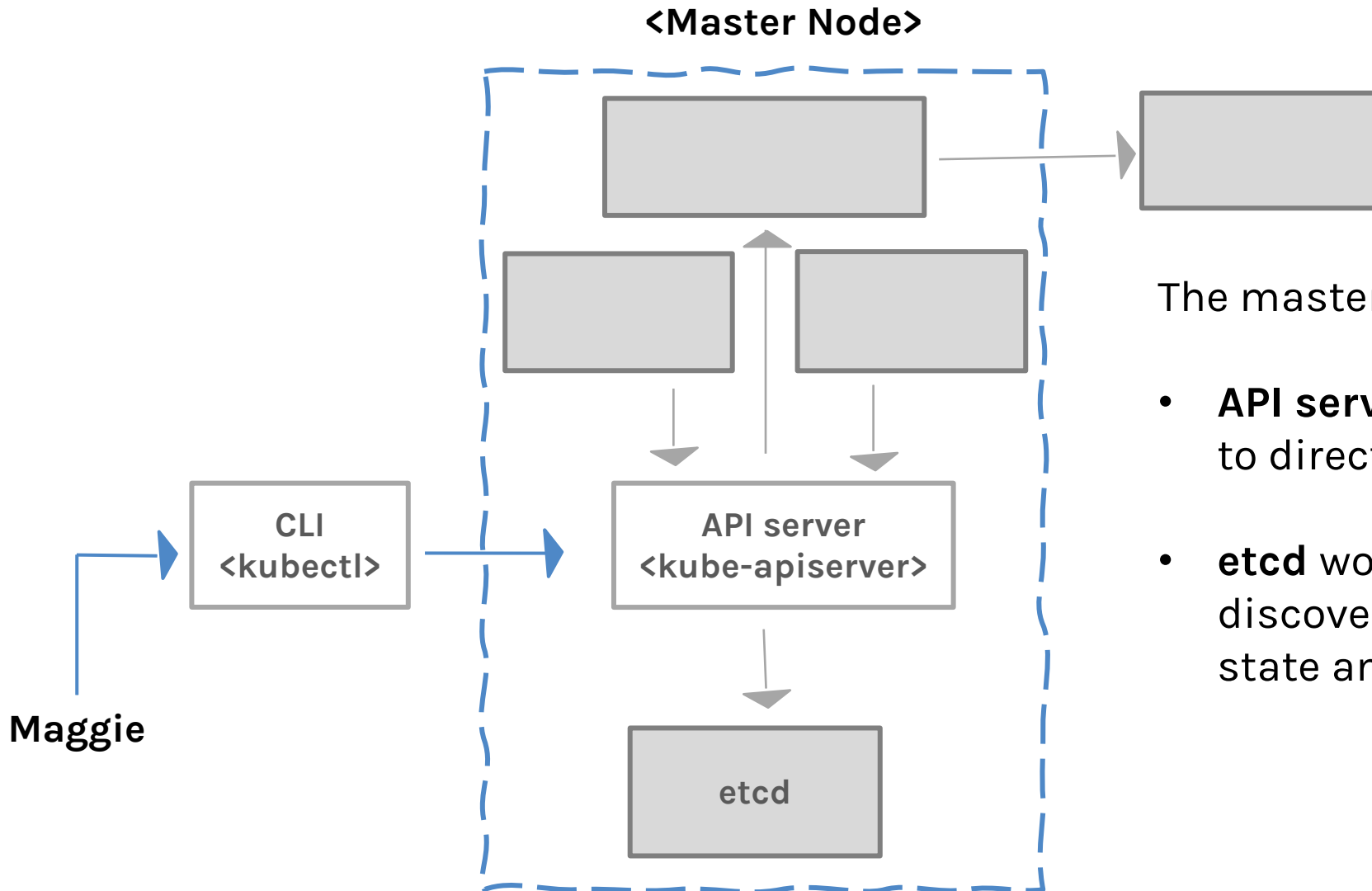
---

Maggie is going to develop a cool application for AC295. She decided to use K8s to build the Online Store (?) Application



The master server consists of various components including a

# How do we build it with K8s?

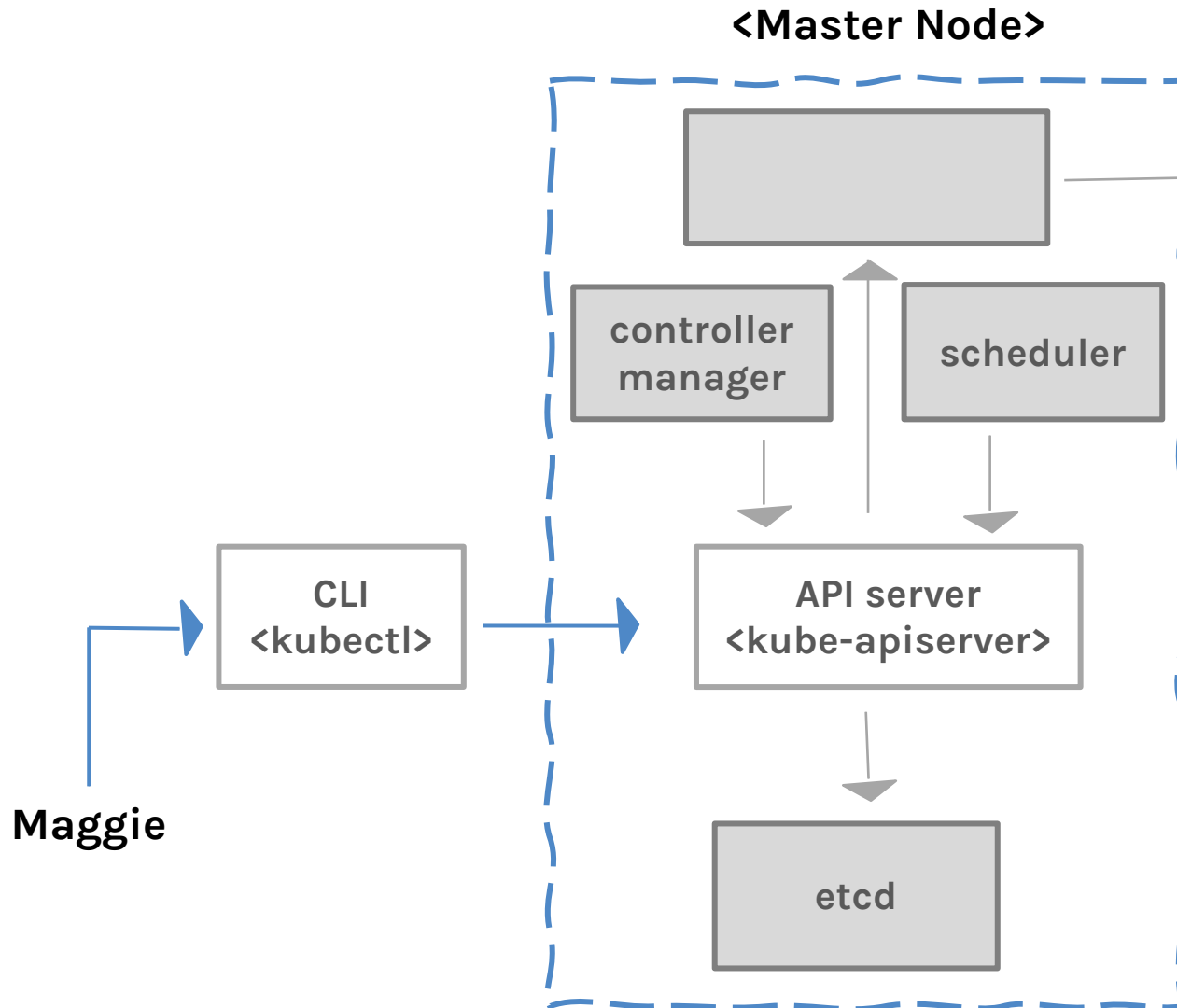


The master node has:

- **API server** contains various methods to directly access the Kubernetes
- **etcd** works as backend for service discovery that stores the cluster's state and its configuration

To learn more  
click [here](#)

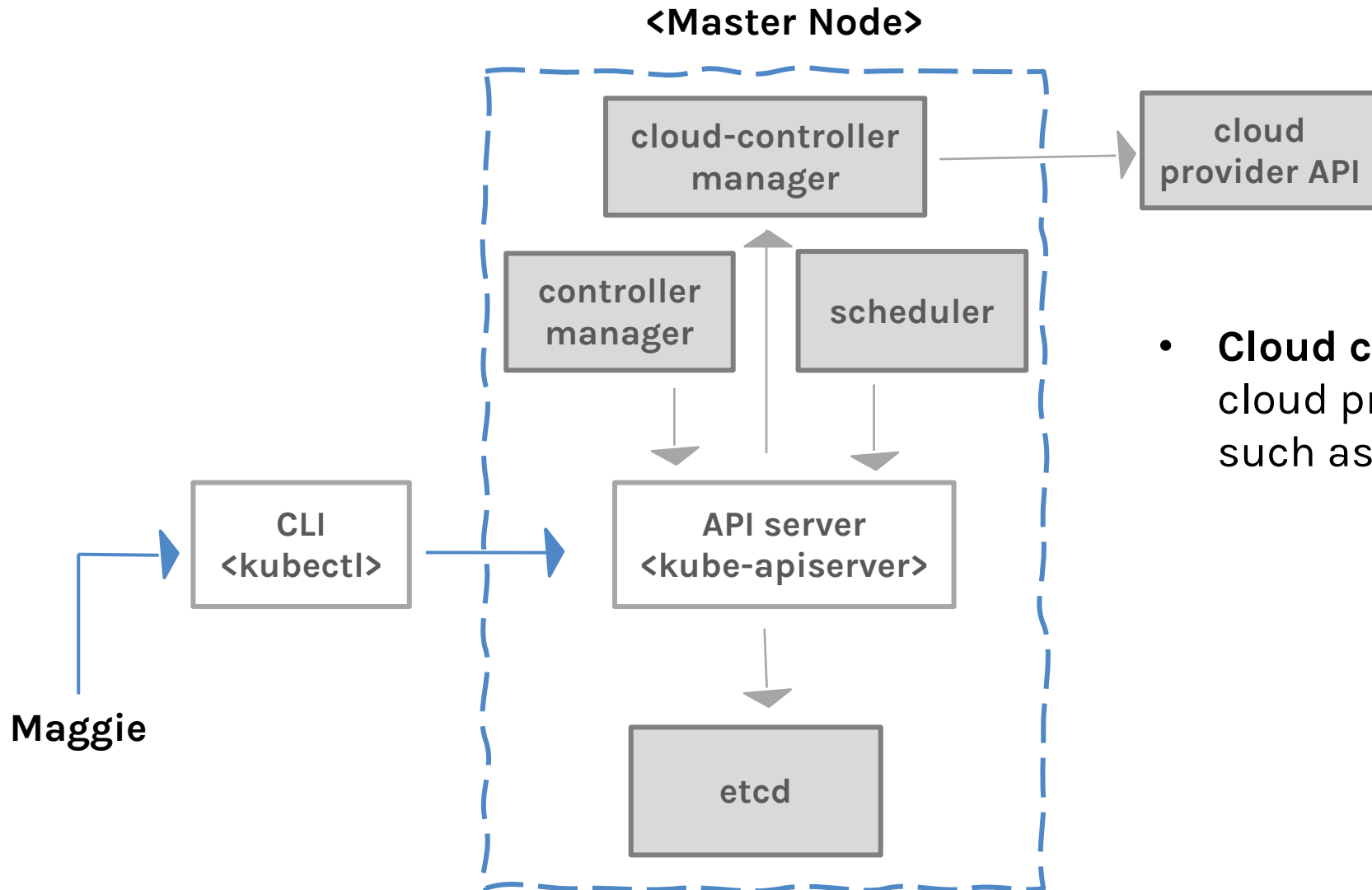
# How do we build it with K8s?



- **Scheduler** assigns to each worker node an application
- **Controller manager:**
  - Keeps track of worker nodes
  - Handles node failures and replicates if needed
  - Provide endpoints to access the application from the outside world

To learn more  
click [here](#)

# How do we build it with K8s?



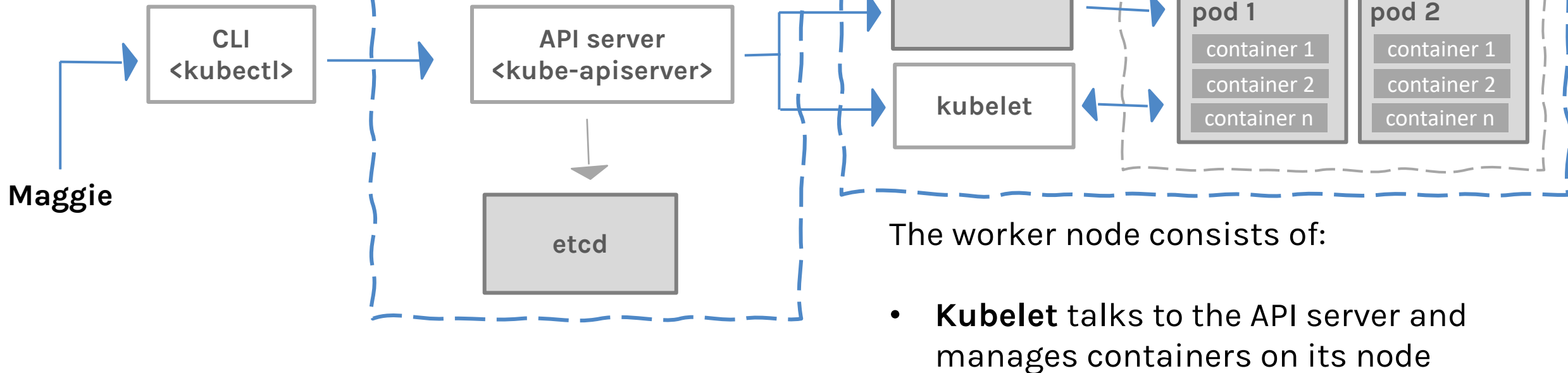
- **Cloud controller** communicates with cloud provide regarding resources such as nodes and IP addresses

To learn more  
click [here](#)

# How do we build it with K8s?

Maggie is going to develop a cool application for AC295. She decided to use **K8s** to build the Online Store (?) Application

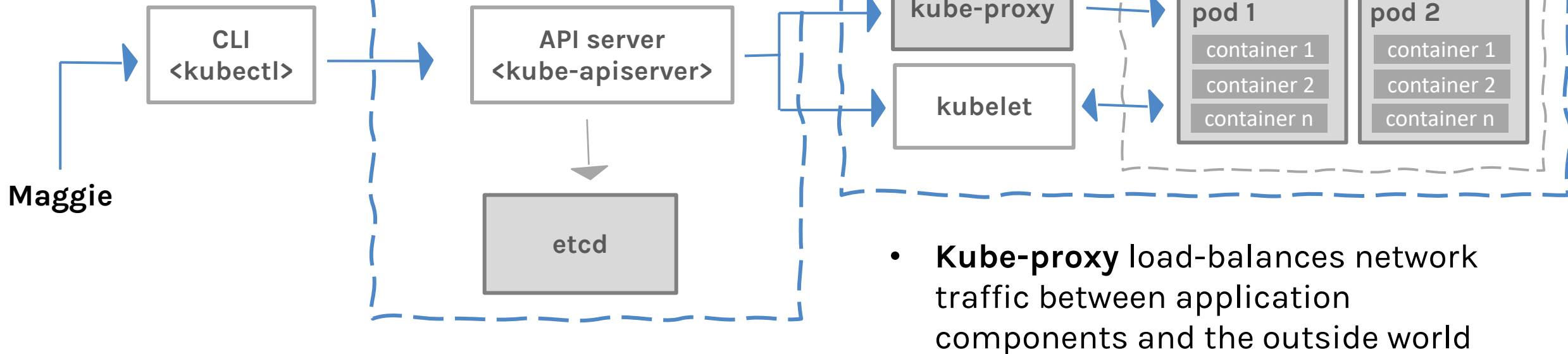
To learn more  
click [here](#)



# How do we build it with K8s?

Maggie is going to develop a cool application for AC295. She decided to use **K8s** to build the Online Store (?) Application

To learn more  
click [here](#)

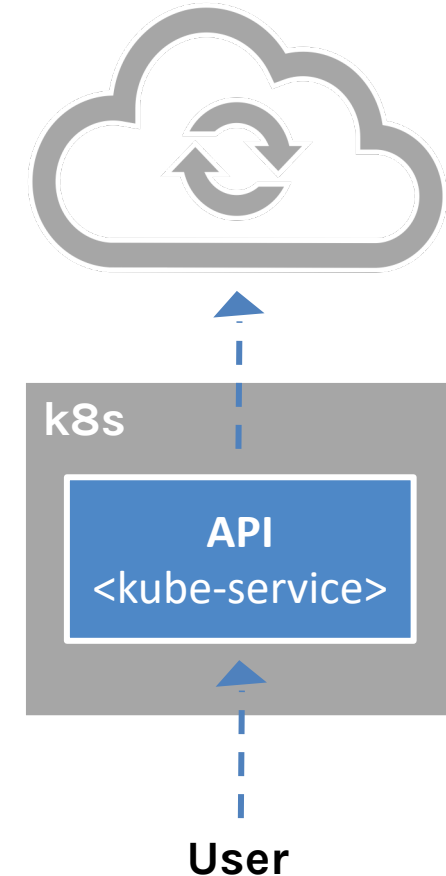




# Advantages of using Kubernetes

There are many reasons why people come to use containers and container APIs like Kubernetes:

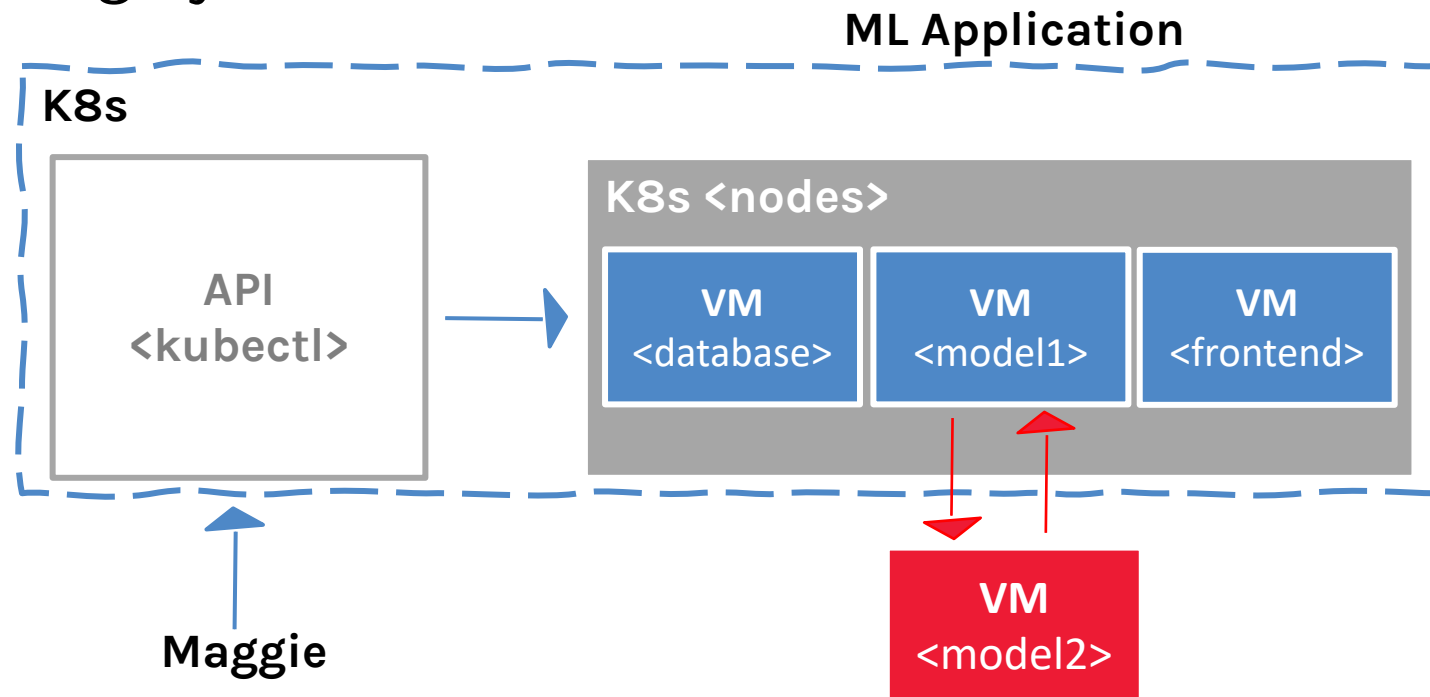
- Velocity
- Scaling (of both software and teams)
- Abstracting the infrastructure
- Efficiency



# Advantages of using Kubernetes: Velocity

It is the speed with which you can respond to innovations developed by others (e.g. change in software industry from shipping CDs to delivering over the network)

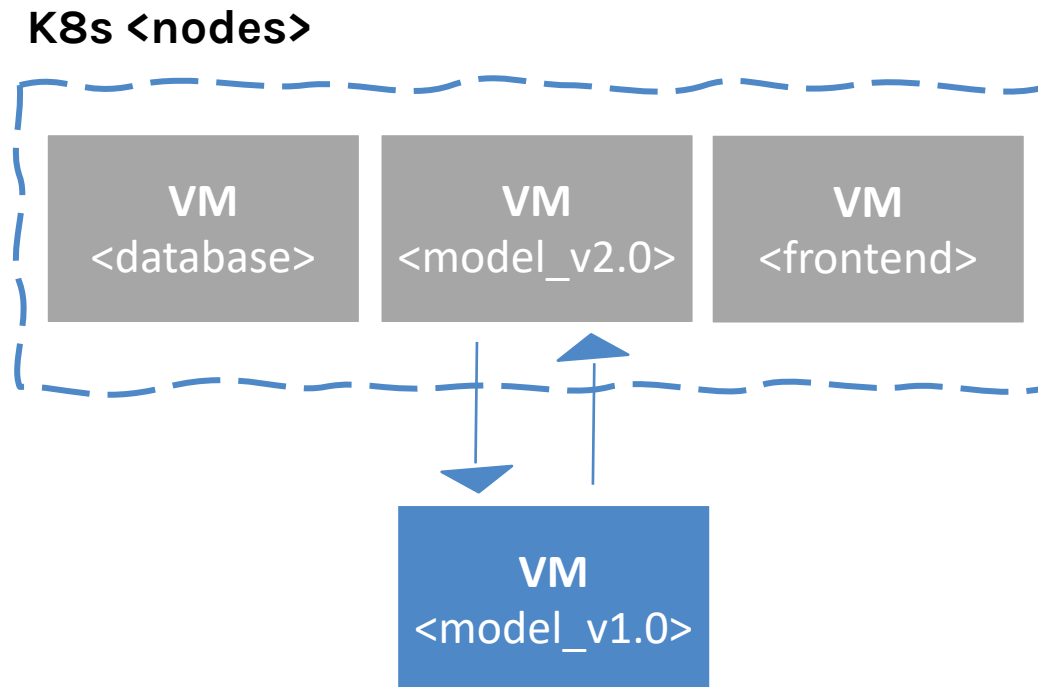
Velocity is measured not in terms of the number of things you can ship while maintaining a highly available service



# Velocity <cont>

Velocity is enabled by:

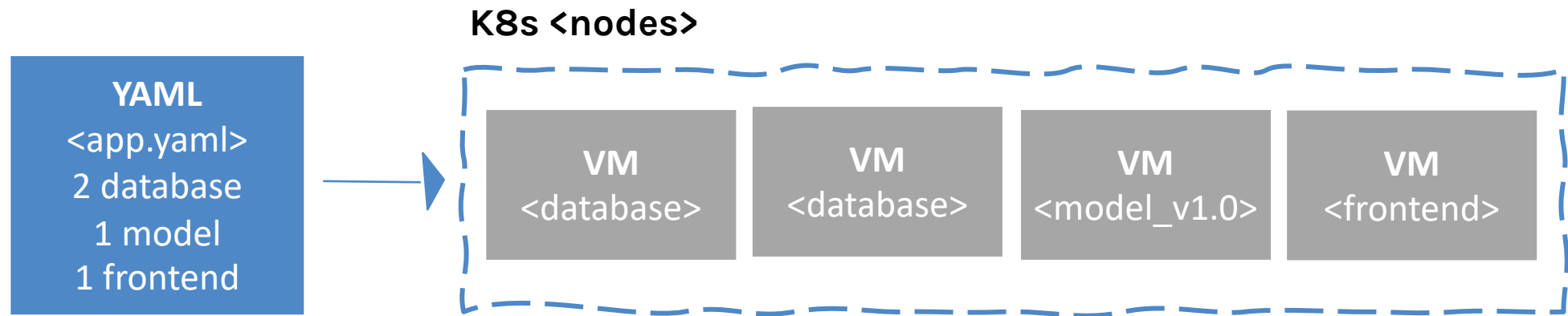
- **Immutable system:** you can't change running container, but you create a new one and replace it in case of failure (allows for keeping track of the history and load older images)



# Velocity <cont>

Velocity is enabled by:

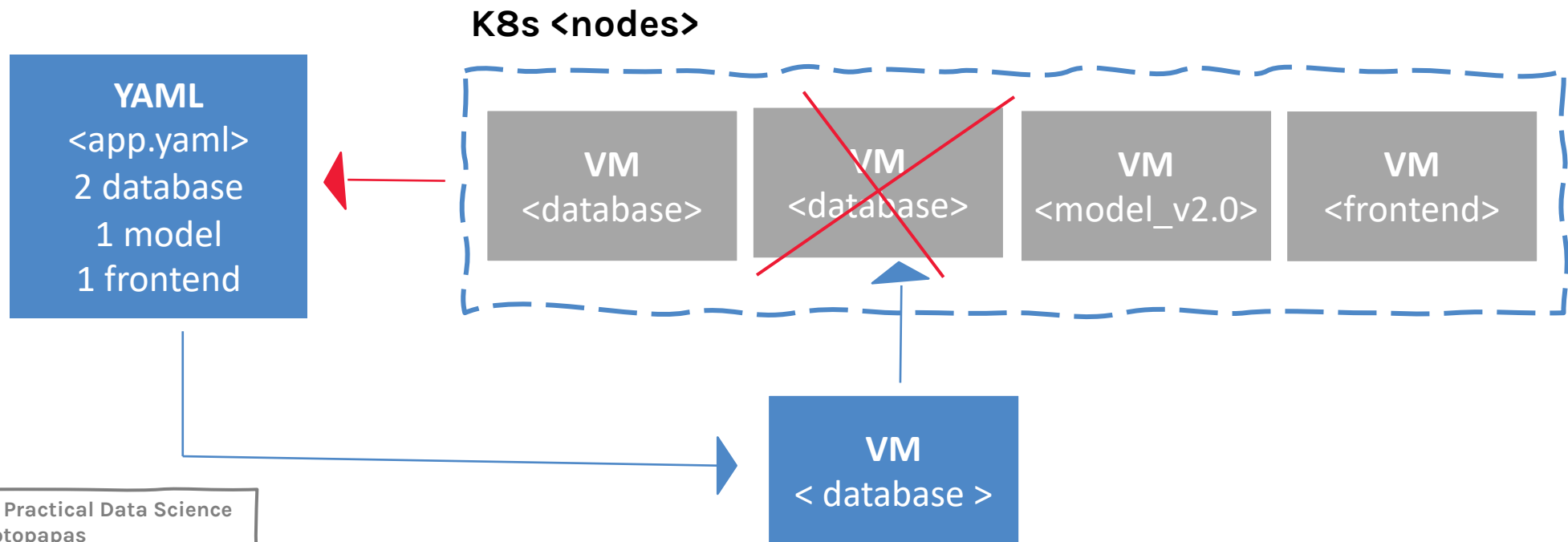
- **Declarative configuration:** you can define the desired state of the system restating the previous declarative state to go back. *Imperative* configuration are defined by the execution of a series of instructions, but not the other way around.



# Velocity <cont>

Velocity is enabled by:

- **Online self-healing systems:** k8s takes actions to ensure that the current state matches the desired state (as opposed to an operator enacting the repair)



# Velocity <recap>

---

Velocity is enabled by:

- Immutable system
- Declarative configuration
- Online self-healing systems

**All these aspects relate to each other to speed up process that can reliably deploy software.**

# Advantages of using Kubernetes: Scaling



As your product grows, it's inevitable that you will need to scale:

- Software
- Team/s that develop it

# Scaling

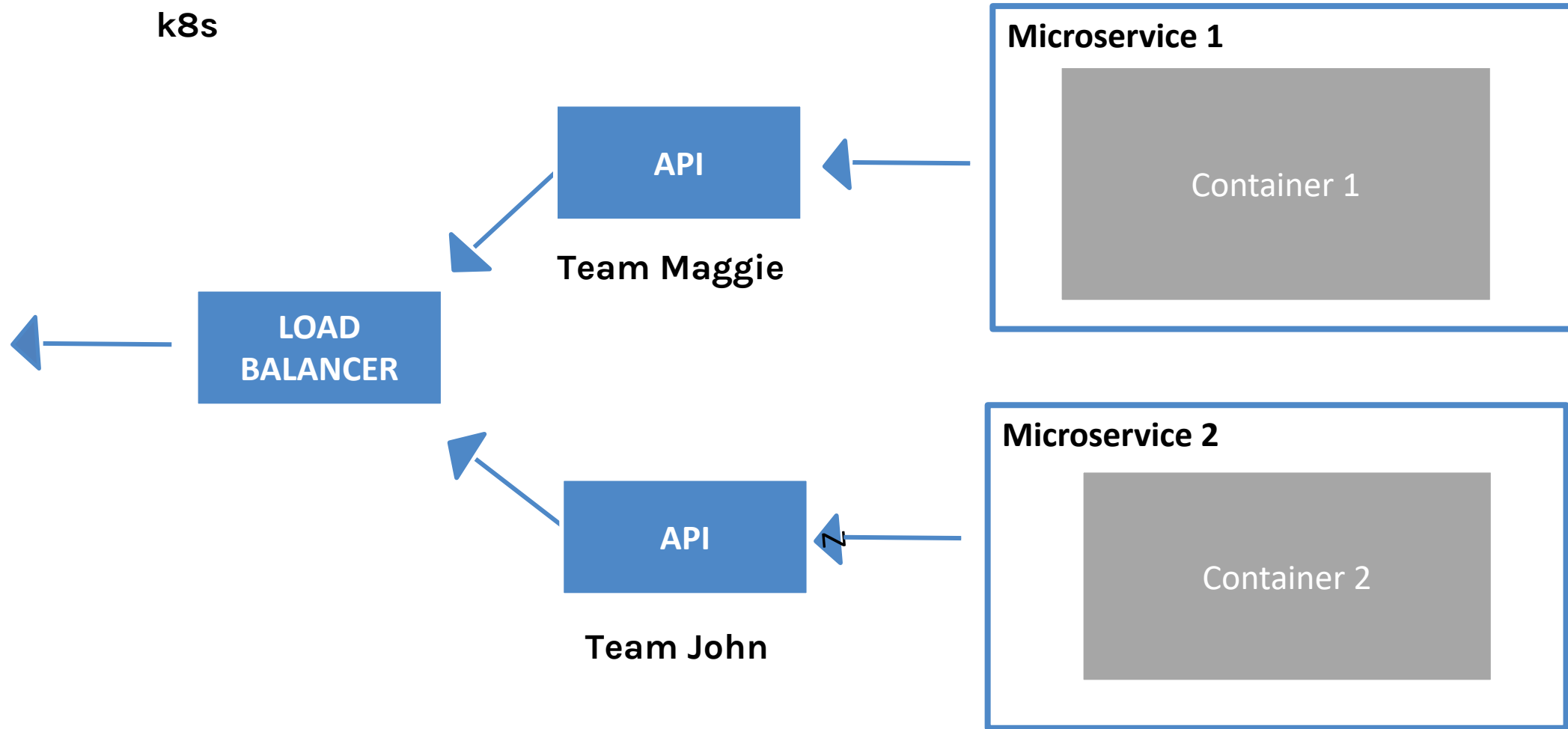
---

**Kubernetes** provides numerous advantages to address scaling:

- **Decoupled architectures:** each component is separated from other components by defined APIs and service load balancers.
- **Easy scaling for applications and clusters:** simply changing a number in a configuration file, k8s takes care of the rest (part of declarative).
- **Scaling development teams with microservices:** small team is responsible for the design and delivery of a service that is consumed by other small teams (optimal group size: 2 pizzas team).



# Scaling <cont>



# Scaling <cont>

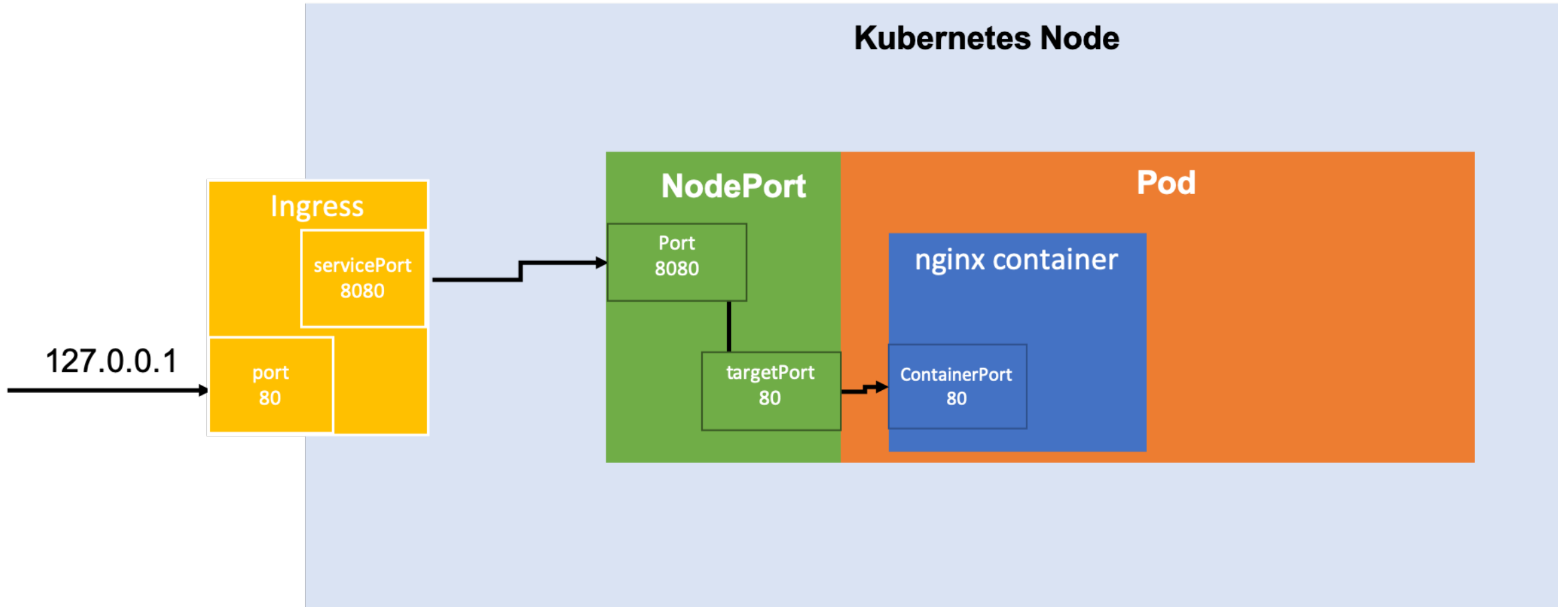
---

**Kubernetes** provides numerous **abstractions** and APIs that help building these decoupled microservice architectures:

- **Pods** can group together container images developed by different teams into a single deployable unit (similar to docker-compose)
- **Other services to isolate** one microservice from another such (e.g. load balancing, naming, and discovery)
- **Namespaces** control the interaction among services
- **Ingress** combine multiple microservices into a single externalized API (easy-to-use frontend)

K8s provides full spectrum of solutions between doing it “the hard way” and a fully managed service

# Scaling <cont>



# Abstracting your infrastructure

---



Kubernetes allows to build, deploy, and manage your application in a way that is portable across a wide variety of environments. The move to application-oriented container APIs like Kubernetes has two concrete benefits:

- **separation:** developers from specific machines
- **portability:** simply a matter of sending the declarative config to a new cluster

# Efficiency

---

There are concrete economic benefit to the abstraction because tasks from multiple users can be packed tightly onto fewer machines:

- **Consume less energy** (ratio of the useful to the total amount)
- **Limit costs of running a server** (power usage, cooling requirements, datacenter space, and raw compute power)
- **Create quickly a developer's test environment** as a set of containers
- **Reduce cost of development instances in your stack**, liberating resources to develop others that were cost-prohibitive

# Outline

---

1: Communications

2: Recap

3: Introduction to Kubernetes

**4: Creating and Running Containers | Review**

5: Anatomy of a Kubernetes Cluster

6: Deploying a Kubernetes Cluster

7: Common kubectl Commands

# Creating and Running Containers | Review

We have already seen how to package an application using the Docker image format and how to start an application using the Docker container runtime:

- We discussed what containers are and what you should use them
- How to build images and update an existing image using Docker (i.e. Dockerfile)
- How to store images in a remote registry (i.e. tag and push to DockerHub)
- How to run container with Docker (generally in Kubernetes containers are launched by a daemon on each node called the kubelet)

# Outline

---

1: Communications

2: Recap

3: Introduction to Kubernetes

4: Creating and Running Containers | Review

**5: Anatomy of a Kubernetes Cluster**

6: Deploying a Kubernetes Cluster

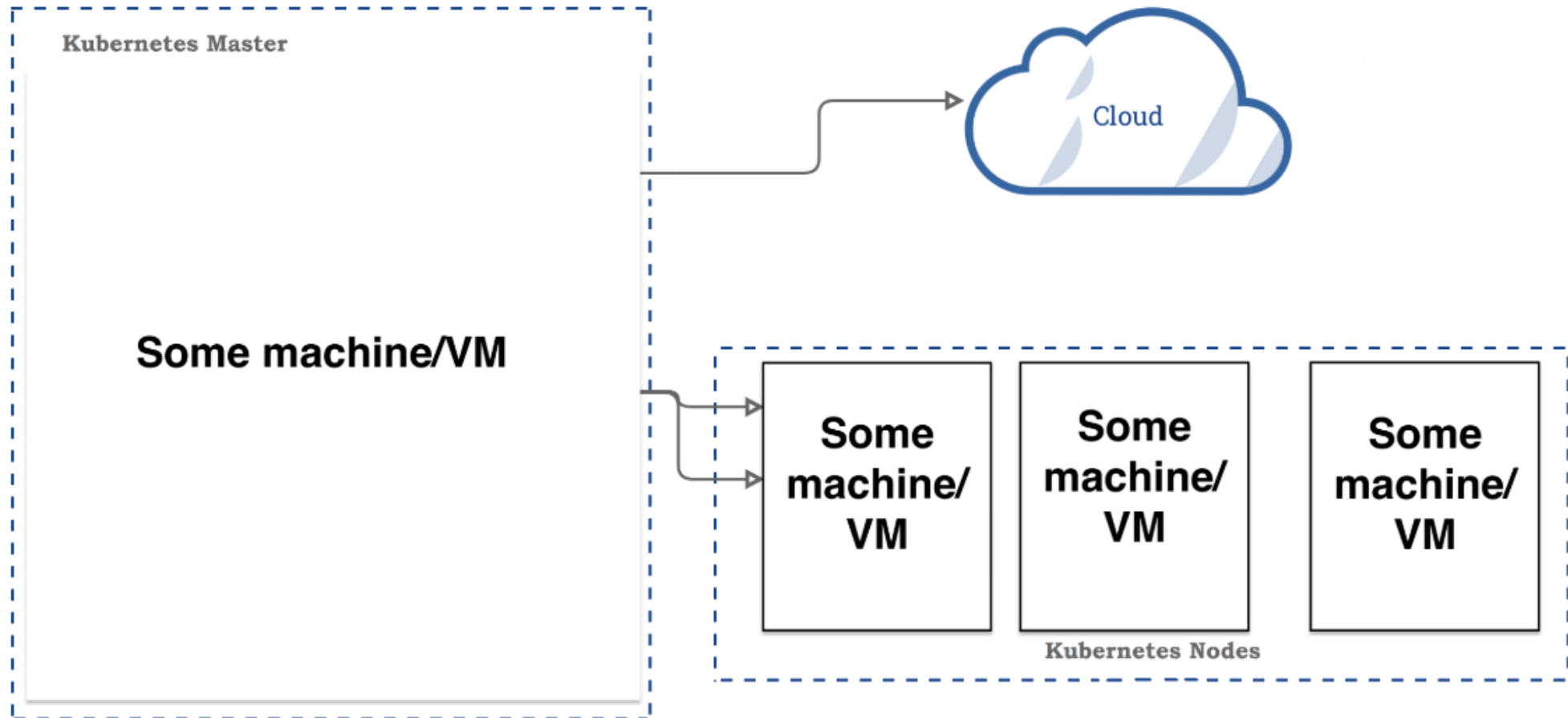
7: Common kubectl Commands



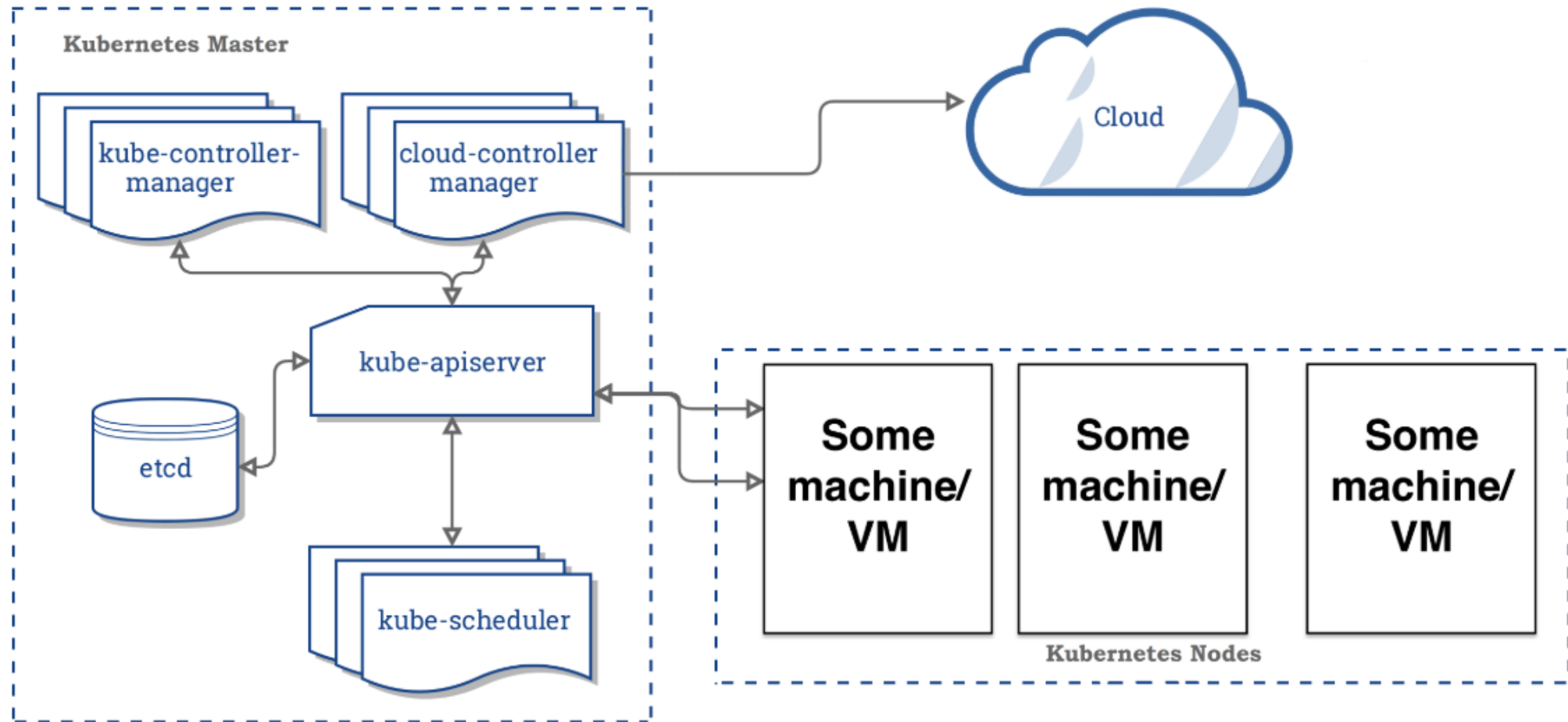
# Anatomy of Kubernetes Cluster

- K8s works on a cluster of machines/nodes
- This could be VMs on your local machine or a group of machines through a cloud provider
- The cluster includes one master node and at least one worker node

# Anatomy of Kubernetes Cluster <cont>



# Anatomy of Kubernetes Cluster | Master Node



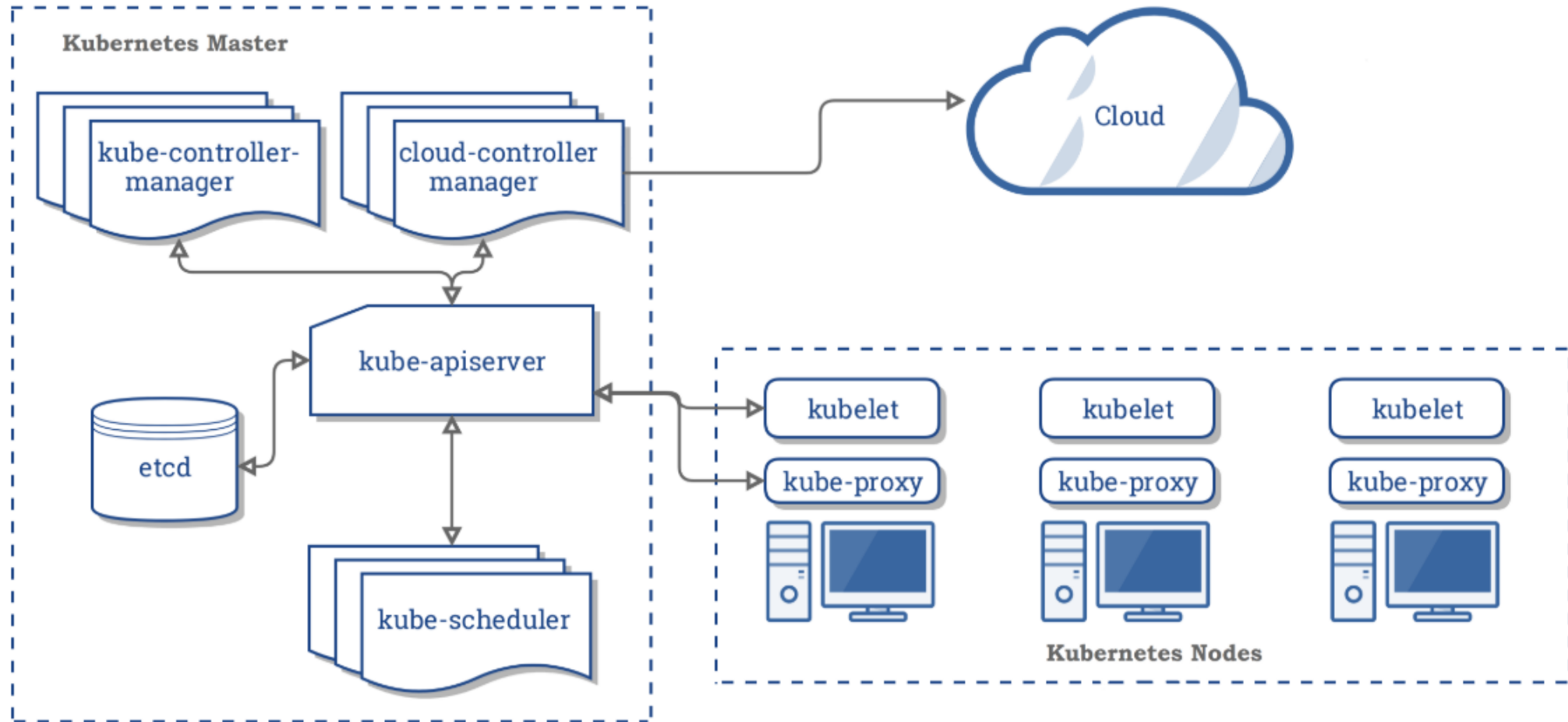
# Anatomy of Kubernetes Cluster | Master Node

**Master node main task** is to manage the worker node(s) to run an application

The master node consists of:

- 1) **API server** contains various methods to directly access the Kubernetes
- 2) **Scheduler** assigns to each worker node an application
- 3) **Controller manager**
  - 3a) Keeps track of worker nodes
  - 3b) Handles node failures and replicates if needed
  - 3c) Provide endpoints to access the application from the outside world
- 4) **Cloud controller** communicates with cloud provide regarding resources such as nodes and IP addresses
- 5) **Etcd** works as backend for service discovery that stores the cluster's state and its configuration

# Anatomy of Kubernetes Cluster | Worker Nodes



# Anatomy of Kubernetes Cluster | Worker Nodes

A worker node consists of:

- 1) **Container runtime** that pulls a specified Docker image and deploys it on a worker node
- 2) **Kubelet** talks to the API server and manages containers on its node
- 3) **Kube-proxy** load-balances network traffic between application components and the outside world

# Outline

---

1: Communications

2: Recap

3: Introduction to Kubernetes

4: Creating and Running Containers | Review

5: Anatomy of a Kubernetes Cluster

**6: Deploying a Kubernetes Cluster**

7: Common kubectl Commands

# Deploying a Kubernetes Cluster

To deploy your cluster you must install Kubernetes. In the exercise you are going to use minikube to deploy a cluster in local mode.

- After installing minikube, use *start* to begin your session creating a virtual machine, *stop* to interrupt it, and *delete* to remove the VM. Below are the commands to execute these tasks:

```
$ minikube start  
$ minikube stop  
$ minikube delete
```



# Deploying a Kubernetes Cluster

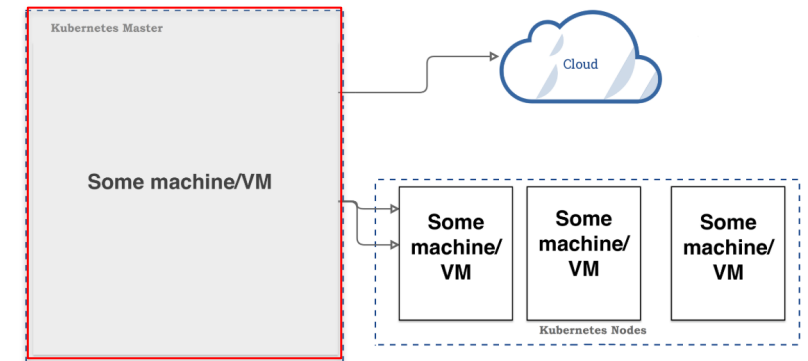
You can easily access the Kubernetes Client using the following command:

- to check your cluster status use:

```
$ kubectl get componentstatuses
```

- and should see output below:

NAME	STATUS	MESSAGE	ERROR
scheduler	Healthy	ok	
controller-manager	Healthy	ok	
etcd-0	Healthy	{"health": "true"}	



# Deploying a Kubernetes Cluster

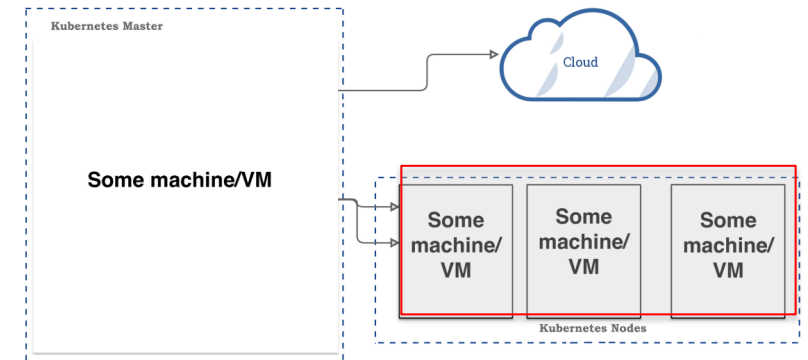
You can easily access the Kubernetes Client using the following command:

- to list the nodes in your cluster use:

```
$ kubectl get nodes
```

- and should see output below:

NAME	STATUS	AGE	VERSION
kubernetes	Ready,master	45d	v1.12.1
node-1	Ready	45d	v1.12.1
node-2	Ready	45d	v1.12.1
node-3	Ready	45d	v1.12.1



# Outline

---

1: Communications

2: Recap

3: Introduction to Kubernetes

4: Creating and Running Containers | Review

5: Anatomy of a Kubernetes Cluster

6: Deploying a Kubernetes Cluster

**7: Common kubectl Commands**

# Common kubectl Commands

Let's practice Kubernetes! Access the exercise using the link below:

> [LINK TO EXERCISE](#) <

> [LINK TO RESOURCES](#) <

# Common kubectl Commands

- Useful commands to complete the exercise:

```
$ kubectl create -f app-db-deploymnet.yaml
$ kubectl get deployment
$ kubectl get pods
$ kubectl get pods /
  -o=custom-columns=NAME:.metadata.name,IP:.status.podIP
$ kubectl create -f app-server-deploymnet.yaml
$ kubectl expose deployment /
app-deployment --type=LoadBalancer --port=8080
$ kubectl get services
$ kubectl delete service app-deployment
$ kubectl delete deployment app-server-deployment
$ kubectl delete deployment app-db-deployment
```

THANK YOU

**AC295**

**Advanced Practical Data Science**  
Pavlos Protopapas