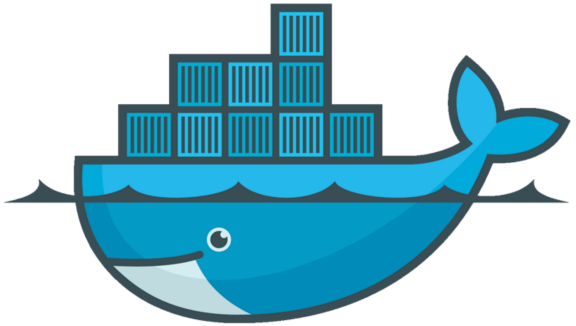
Introduction to Docker and Kubernetes

kubernetes



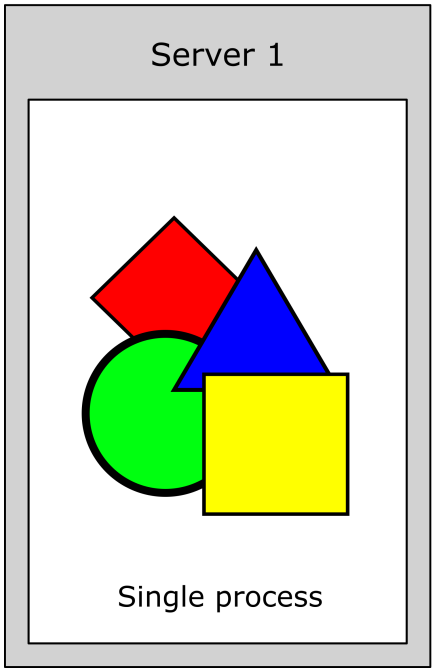
docker

José Joaquín Arias

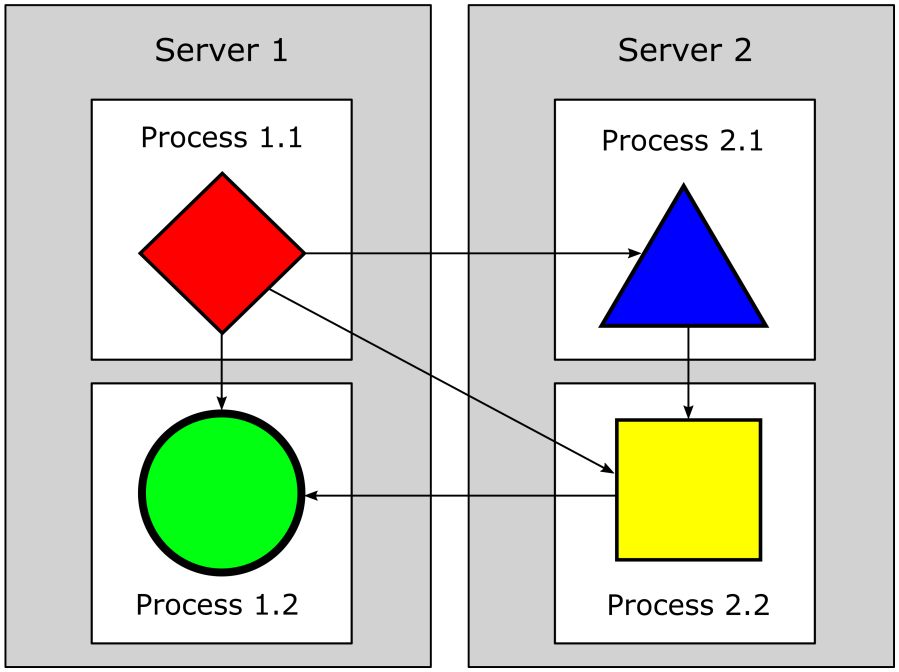
github.com/joaquinariasgomez/Kubernetes-docker

From Monolithic apps to Microservices

Monolithic application



Microservices-based application



Docker

Docker and containers

The idea behind Docker is to create lightweight and portable containers so that apps can run on any machine with Docker installed.

Developers can then not worry about whether their application will work on the machine it will actually run on.

Docker and containers

A container is a single isolated process running in the host OS.

Compared to VMs, containers are much more lightweight, which allows you to run higher number of software components on the same hardware, mainly because each VM needs to run its own set of system processes.

We will use Vagrant as the platform that will provide us with the virtual machine with all the resources to run the tests.

Vagrant is a tool for building and managing virtual machine environments in a single workflow. Machines are provisioned on top of VirtualBox, VMware, AWS or any other provider.

Vagrant will isolate dependencies and their configuration within a single disposable and consistent environment.

First steps with Docker

The following two Docker commands will pull new images from registry and run them with or without arguments.

```
docker run busybox echo "Hello world"
```

```
docker run hello-world
```

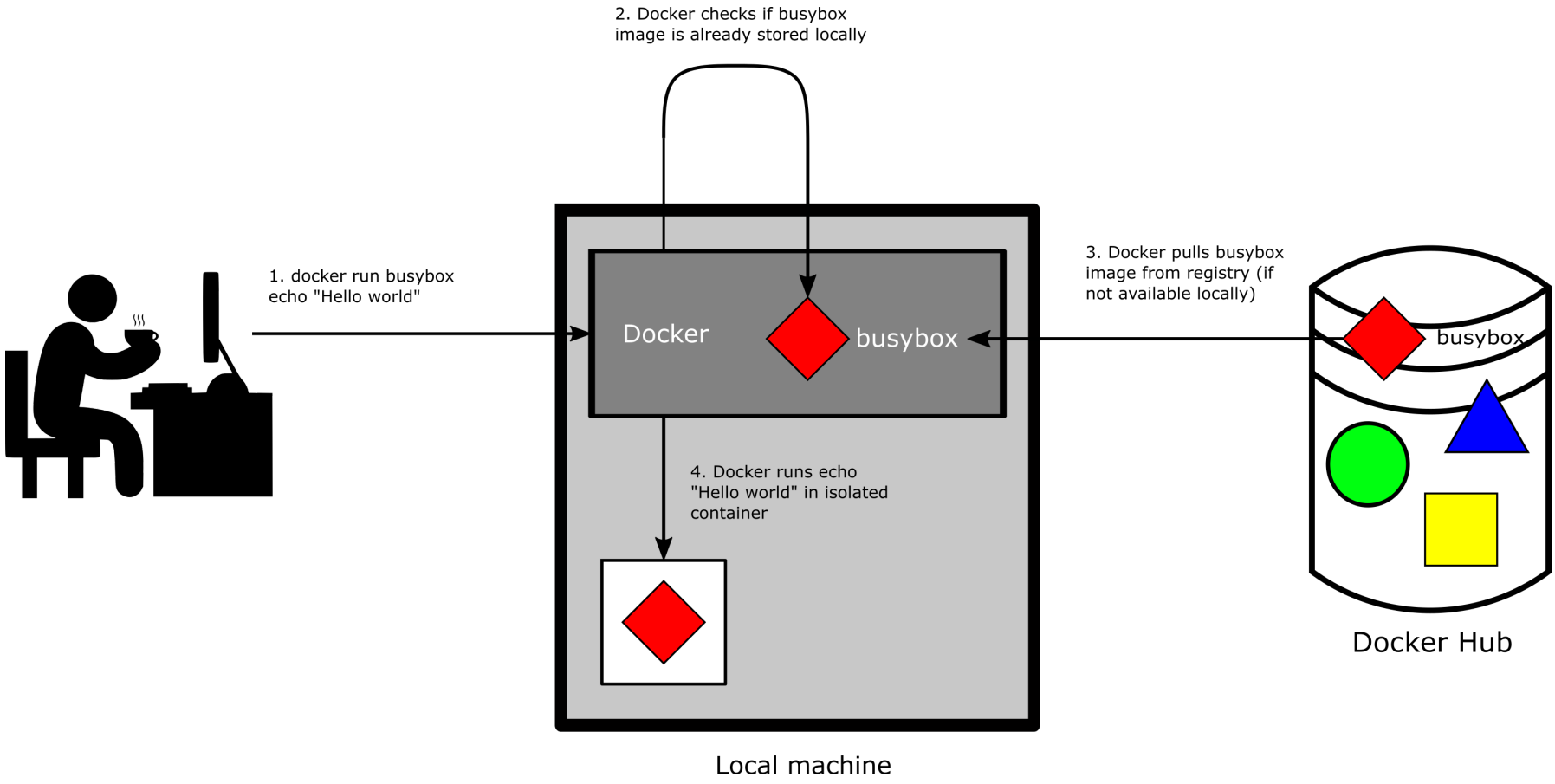
If we run the same Docker command, we will see that it won't pull any image from the registry.

```
docker run hello-world
```

We can show the images stored in our local machine by running the following Docker command.

```
docker images
```

These images are run in a container completely isolated from all other processes running in our virtual machine.


2. Docker checks if busybox
image is already stored locally
1. docker run busybox
echo "Hello world"
Docker
busybox
3. Docker pulls busybox
image from registry (if
not available locally)
busybox
4. Docker runs echo
"Hello world" in isolated
container
Local machine
Docker Hub

To distinguish the version of an image, colon is added after the name and then the version is written.

```
docker run <image>:<tag>
```

If you don't specify the tag, Docker will assume you are referring to <latest>.

Creating a simple app in Node.js

We will create a web app deployed on a container with Docker's help.

This app will accept HTTP requests from any client and will answer with a OK (200 status code) from the server.

App.js

```
const http = require('http');
const os = require('os');

console.log("Server starting...");

var handler = function(request, response) {
  console.log("Received request from "+request.connection.remoteAddress);
  response.writeHead(200);
  response.end("Hello! Hostname of this server: "+os.hostname());
};

var www=http.createServer(handler);
www.listen(8080);
```

Dockerfile

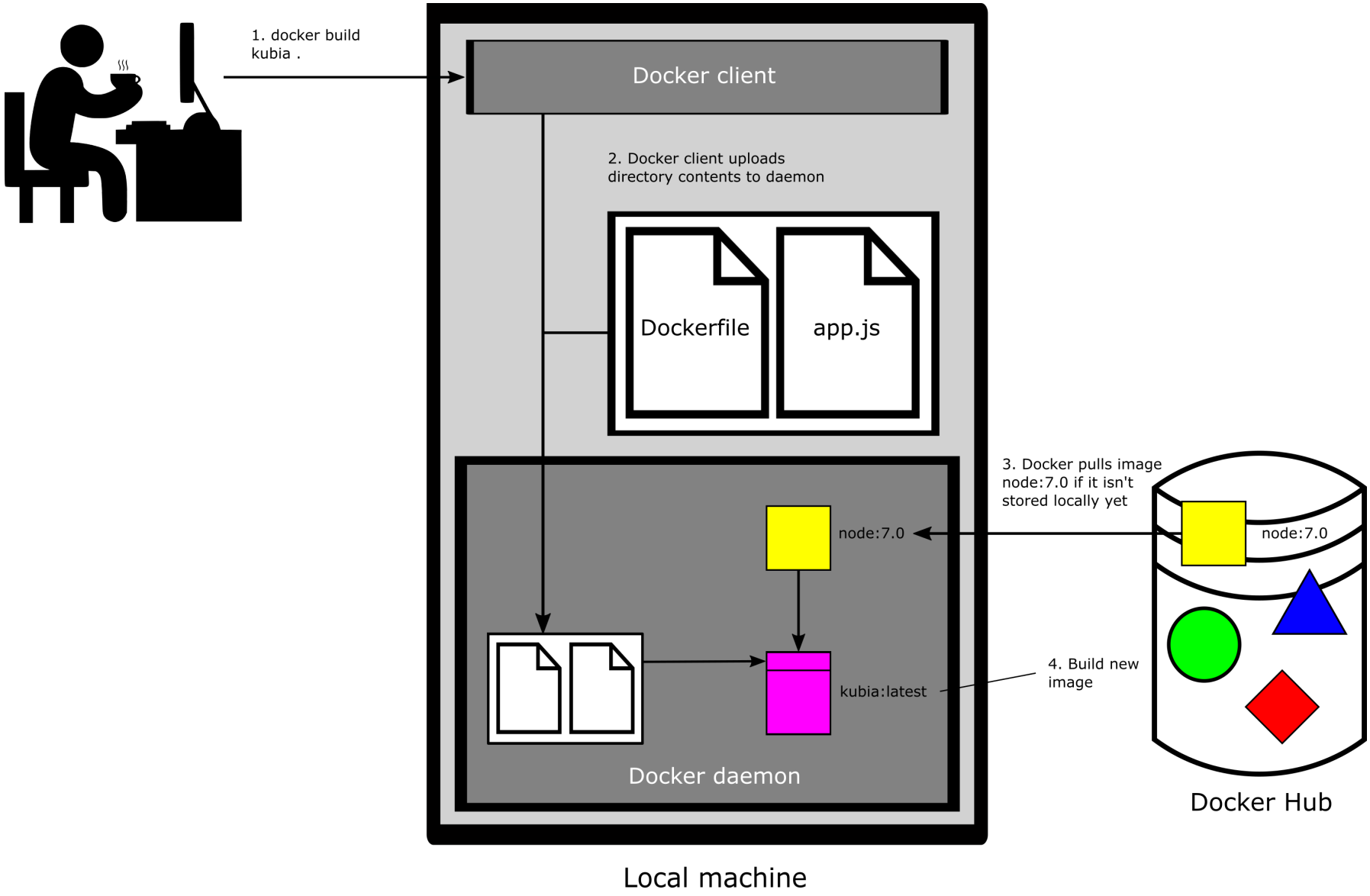
```
FROM node:7
ADD app.js /app.js
ENTRYPOINT ["node", "app.js"]
```

Building the image

Once we have the Dockerfile, we will run the following command to build an image called "kubia".

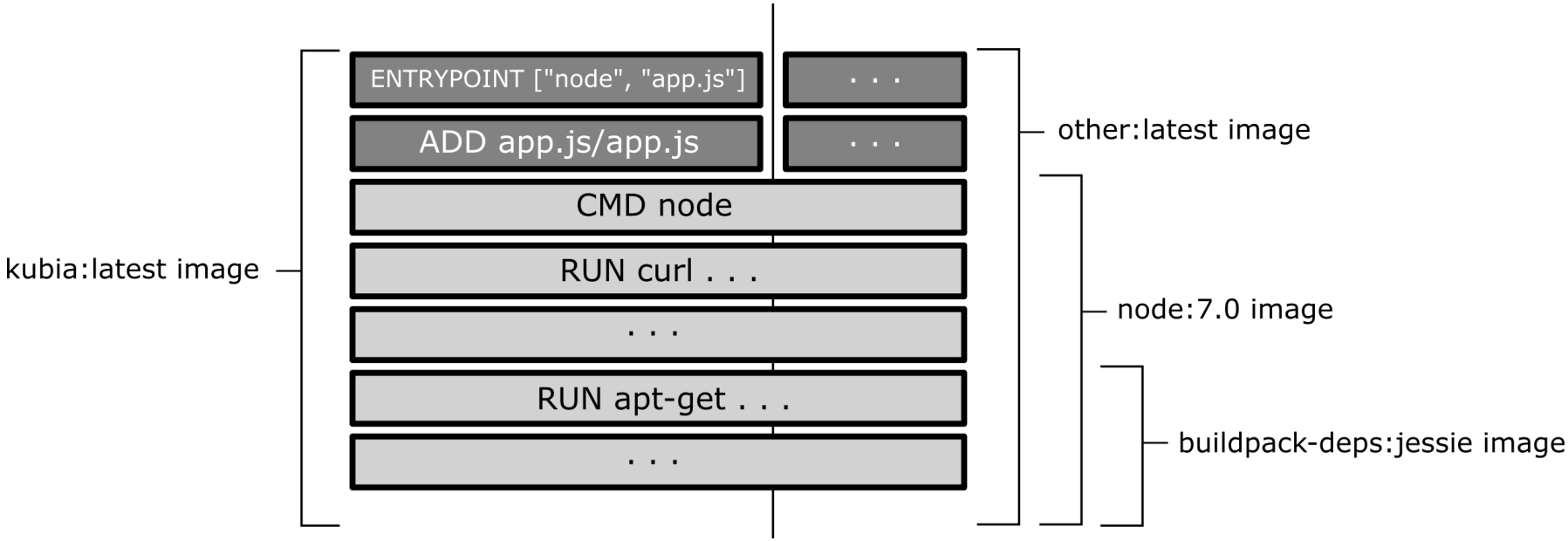
```
docker build -t kubia .
```

Docker will find the Dockerfile in the specified direction ("." in this case).


1. docker build
kubia .
Docker client
2. Docker client uploads
directory contents to daemon
Dockerfile
app.js
node:7.0
kubia:latest
Docker daemon
3. Docker pulls image
node:7.0 if it isn't
stored locally yet
node:7.0
4. Build new
image
Docker Hub
Local machine

An image is composed of several layers, making the pull process more efficient.

Apart from that, the construction of a Docker image creates a new layer for each instruction of the Dockerfile.


ENTRYPOINT ["node", "app.js"]
. . .
ADD app.js/app.js
. . .
CMD node
RUN curl . . .
. . .
RUN apt-get . . .
. . .
kubia:latest image
other:latest image
node:7.0 image
buildpack-deps:jessie image

Now let's run our image with the folllowing command:

```
docker run --name kubia-container -p 8080:8080 -d kubia
```

Once this is done, we can access our application through http://localhost:8080.

```
curl localhost:8080
```

If you are using Docker in Mac or Windows, you are not running the Docker daemon inside the same virtual machine.

You will then need the name or IP of the virtual machine where this Docker daemon is running. To look at it, use the DOCKER_HOST environment variable.

The hexadecimal number we get is the ID of the Docker container.

```
docker ps
```

If we execute the command above we will see a list of the containers Docker is running right now.

Exploring the interior of a container

Since our image is based on Node.js, which has bash, we can now run the following to enter inside it.

docker exec -it kubia-container bash

To stop our running container.

docker stop kubia-container

We can still see a non running container.

docker ps -a

To remove our running container.

docker rm kubia-container

We will use Docker Hub to store our image. To do that, we will have to retag our image according to Docker rules.

docker tag kubia <docker_id>/kubia

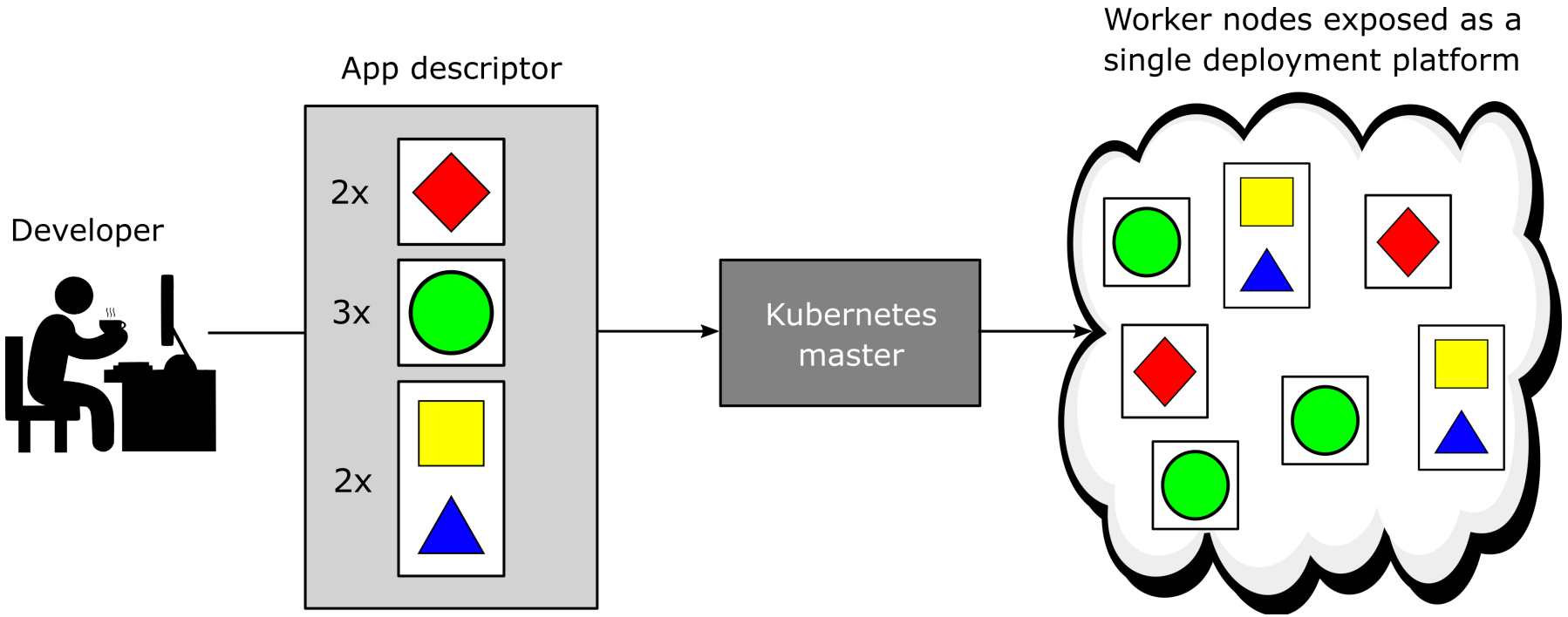
Once we are logged in Docker, we can push our image to the Docker Hub. And once it is pushed, everyone will be able to run it.

```
docker push <docker_id>/kubia
```

```
docker run -p 8080:8080 -d <docker_id>/kubia
```

Kubernetes

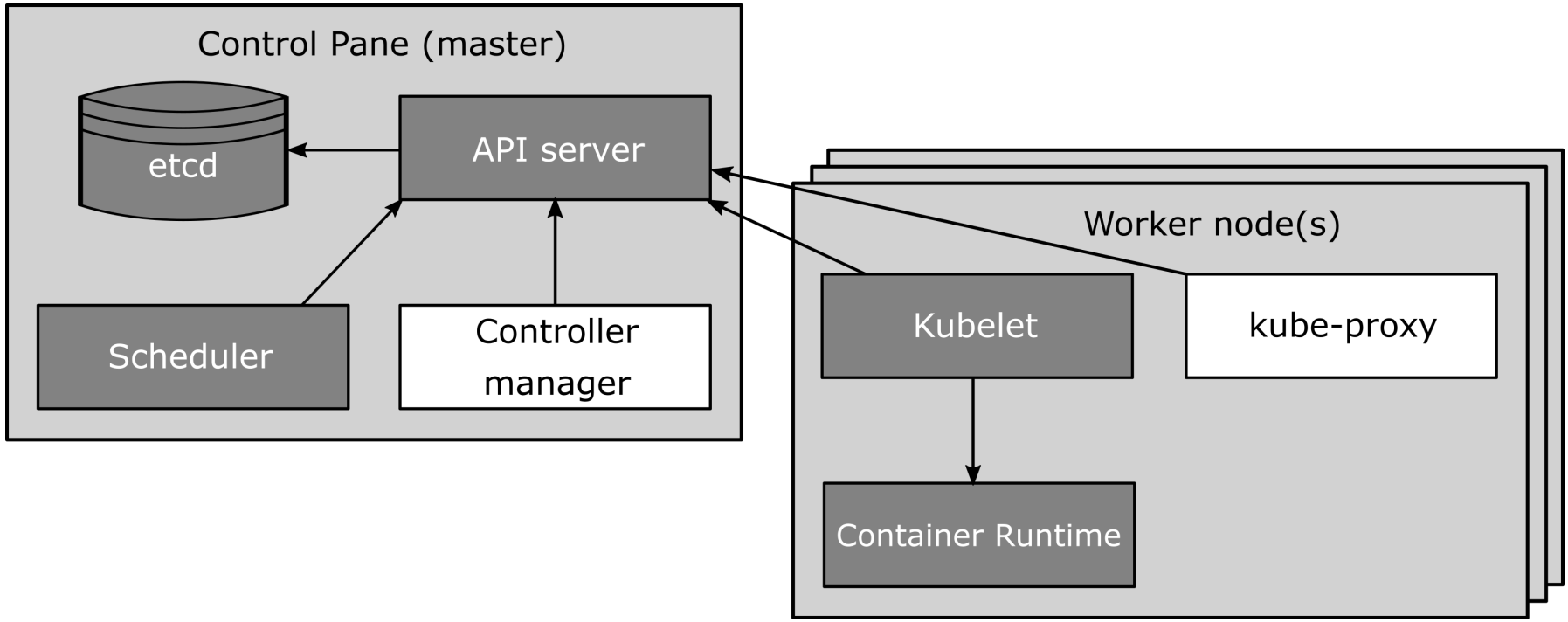
Kubernetes is an open source system created by Google for the management of applications in containers, an orchestration system for Docker containers, allowing actions such as scheduling the deployment, scaling and monitoring of our containers.


Developer
App descriptor
2x
3x
2x
Kubernetes master
Worker nodes exposed as a single deployment platform

Understanding the architecture of a Kubernetes cluster

A Kubernetes cluster is composed of a set of nodes of two types.

- Master node, which contains the Control Plane. Its job is to control the cluster.
- Worker nodes, which will actually run the application.


Control Pane (master)
etcd
API server
Scheduler
Controller manager
Worker node(s)
Kubelet
kube-proxy
Container Runtime

Setting up a cluster

Once our image is packaged using Docker, we can deploy it in a Kubernetes cluster instead of using Docker directly.

The simplest way of setting up a Kubernetes cluster is using Minikube. Running the following command will give you a description about this cluster.

```
kubectl cluster-info
```

The way of running our application using Kubernetes is with a Docker-like command.

```
kubectl run kubia --image=<docker_id>/kubia --port=8080 --generator=run/v1
```

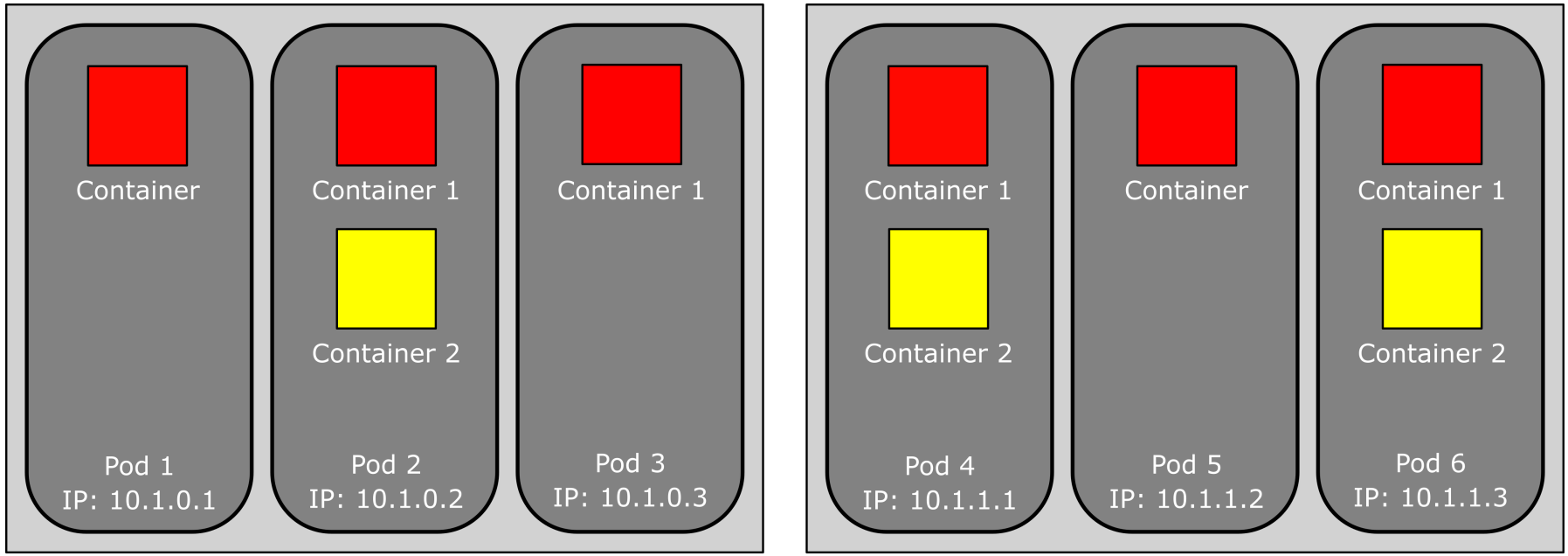
Since we are telling Kubernetes to create a replication controller, we can check that it is present by running:

```
kubectl get replicationcontroller
```

Introducing Pods

The minimum unit that Kubernetes handles is called Pod. It is, essentially, a group of containers.

Each Pod is like an isolated virtual machine. It has its own IP, hostname, processes.. running an application.

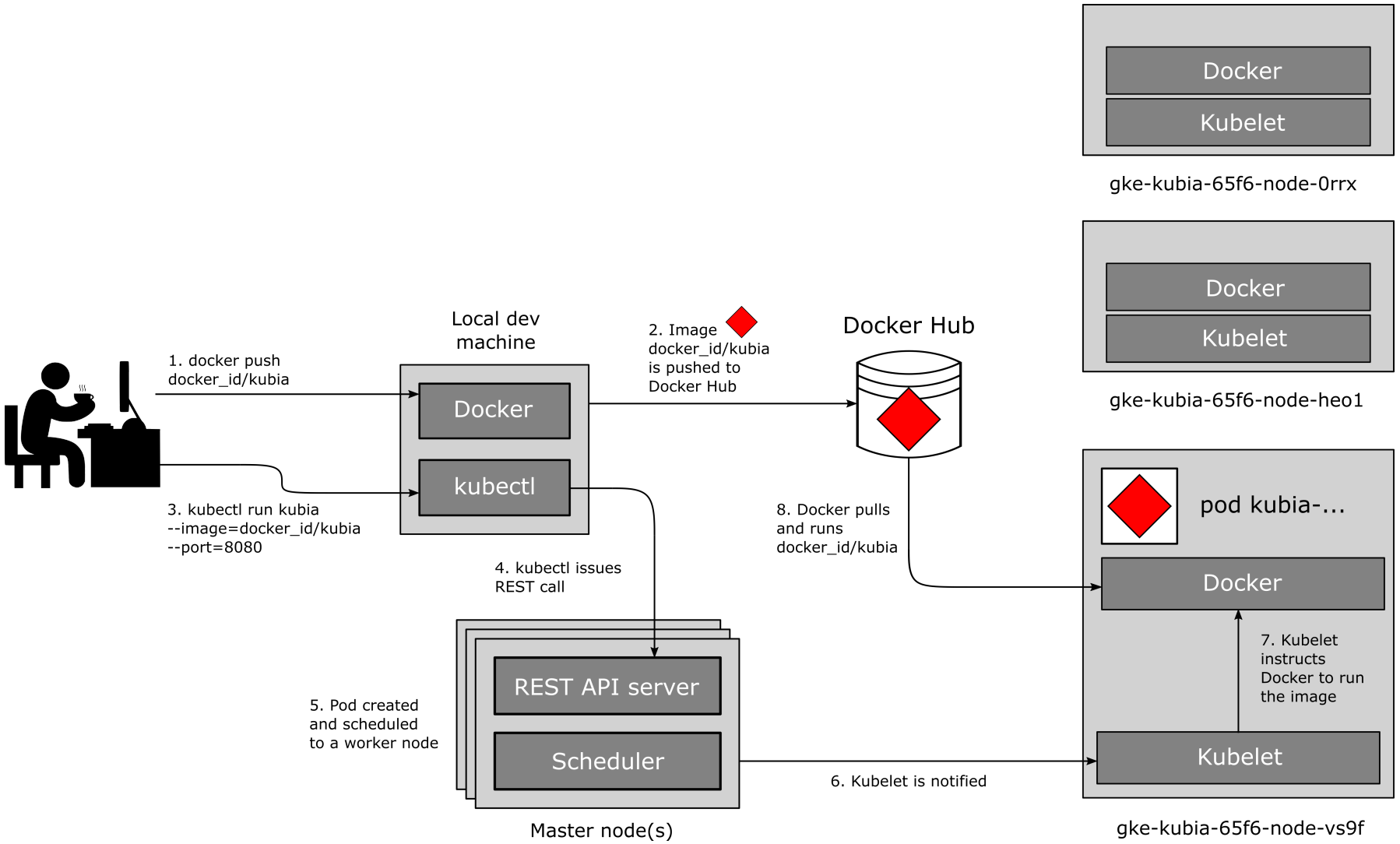

Container
Pod 1
IP: 10.1.0.1
Container 1
Container 2
Pod 2
IP: 10.1.0.2
Container 1
Pod 3
IP: 10.1.0.3
Container 1
Container 2
Pod 4
IP: 10.1.1.1
Container
Pod 5
IP: 10.1.1.2
Container 1
Container 2
Pod 6
IP: 10.1.1.3
Worker node 1
Worker node 2

List all running pods.

```
kubectl get pods
```

Obtain more information about a pod.

```
kubectl describe pod <pod_name>
```


Docker
Kubelet
gke-kubia-65f6-node-0rrx
Docker
Kubelet
gke-kubia-65f6-node-heo1
Local dev machine
1. docker push docker_id/kubia
2. Image docker_id/kubia is pushed to Docker Hub
Docker Hub
Docker
kubectl
3. kubectl run kubia --image=docker_id/kubia --port=8080
4. kubectl issues REST call
8. Docker pulls and runs docker_id/kubia
pod kubia-...
Docker
7. Kubelet instructs Docker to run the image
REST API server
Scheduler
5. Pod created and scheduled to a worker node
6. Kubelet is notified
Kubelet
Master node(s)
gke-kubia-65f6-node-vs9f

Horizontally scaling your application

To scale up the number of replicas of your pod, you need to change the desired replica count on the ReplicationController like this:

kubectl scale rc kubia --replicas=3

Kubernetes Dashboard

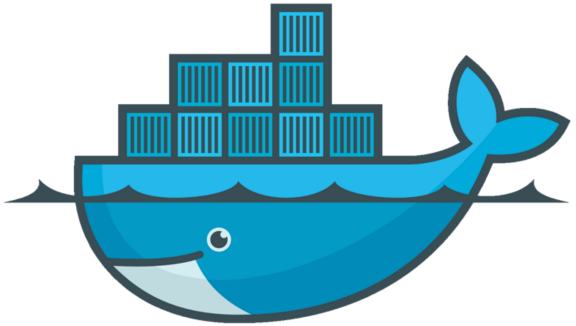
The dashboard allows us to watch all the events that kubectl ran and more, but graphically.

To obtain its private url, we can run the following command.

```
minikube dashboard --url
```

Introduction to Docker and Kubernetes

kubernetes



docker

José Joaquín Arias

github.com/joaquinariasgomez/Kubernetes-docker