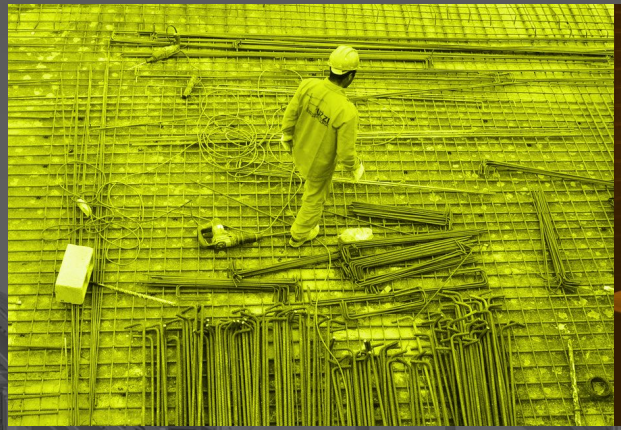
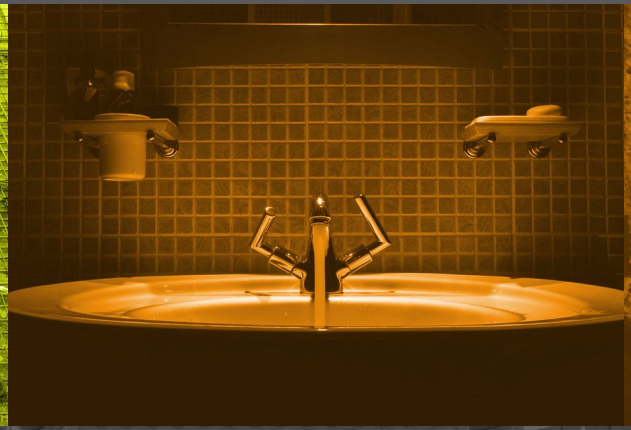
Kubernetes 101 workshop

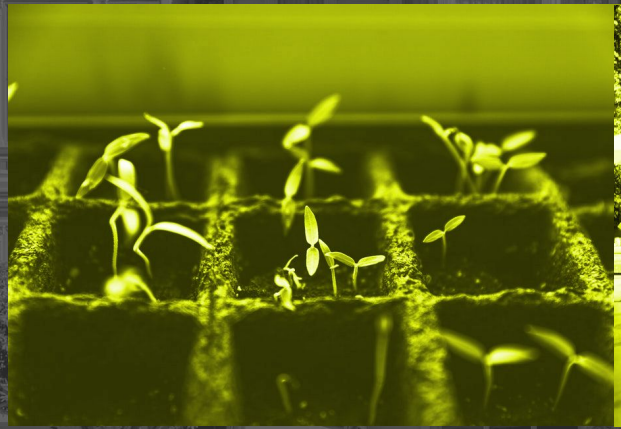
@sathishvj

Virtual Machines and Containers


POLICE

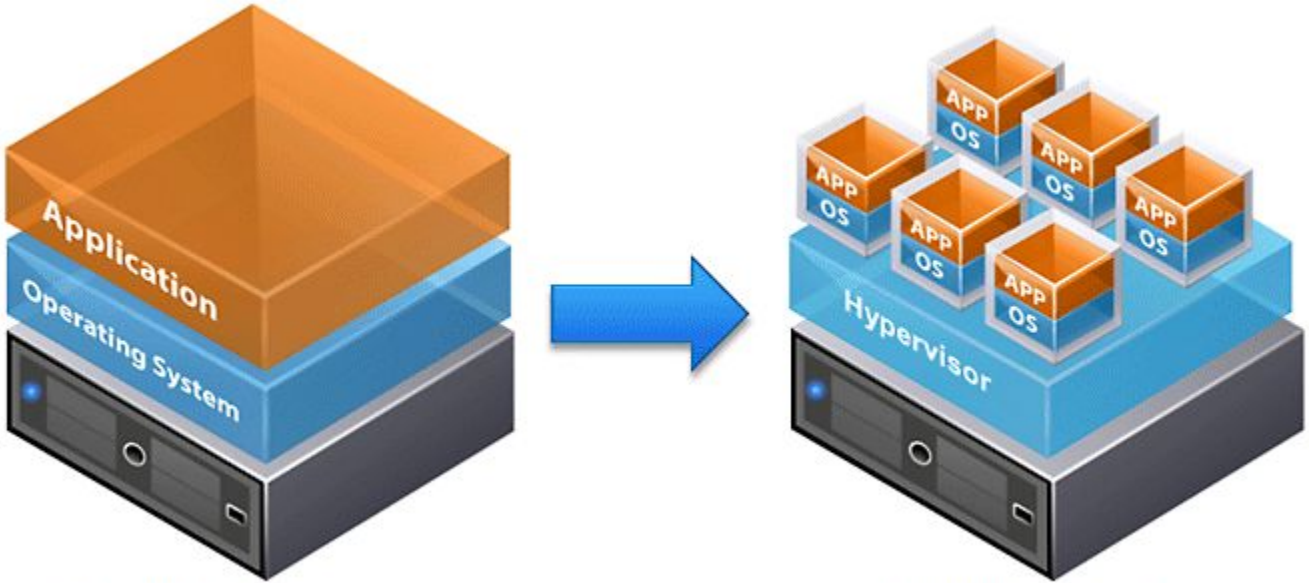







POLICE


DO
MORE.



Traditional Architecture

Virtual Architecture


ORACLE
VirtualBox

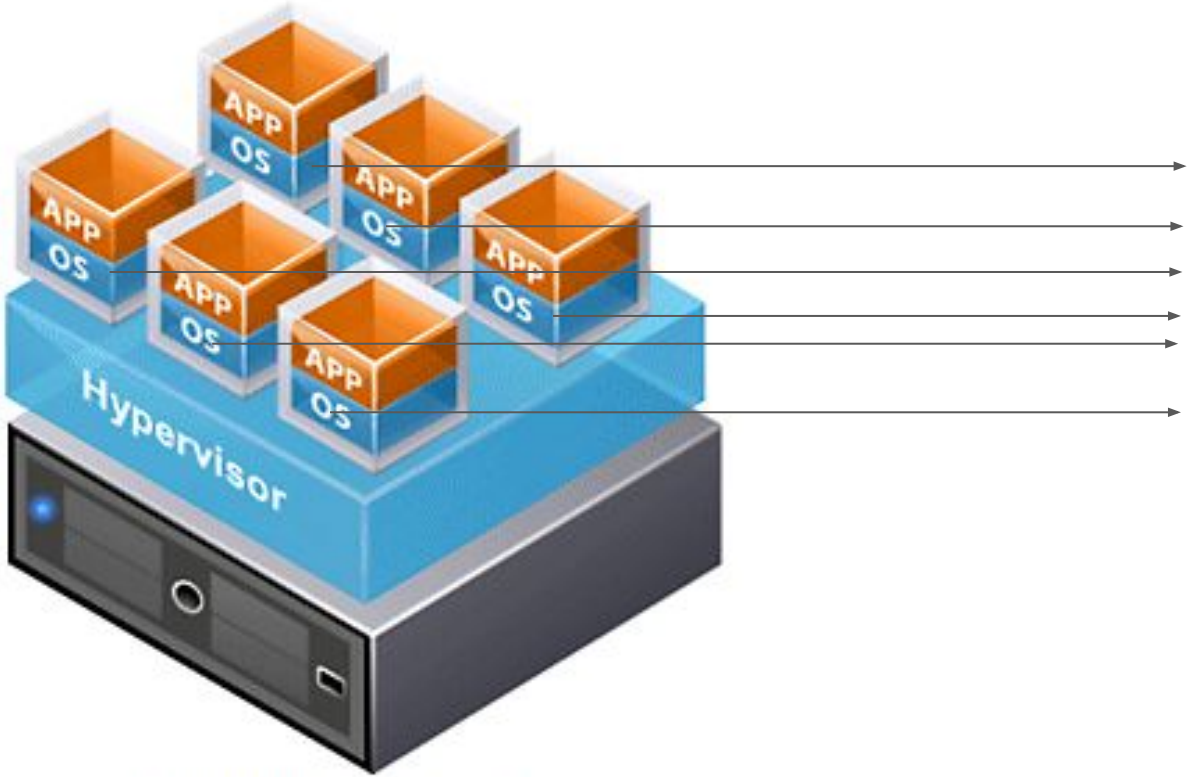
VirtualBox


QEMU


Parallels®


vmware®


Microsoft
Virtual PC



Virtual Architecture

6 x OS


IM JUST SAYIN'
YOU COULD DO
BETTER

Container Ecosystem Evolution Timeline

1979 — Unix v7

Introduction of chroot system call

2000 — FreeBSD jails

Developed by Derrick Woolworth & adopted by Poul-Henning Kamp

2001 — Linux VServer / VServer project

Virtualization of operating systems

2004 — Oracle Solaris Zones

Addons like snapshot and cloning

2005 — Open Virtuzzo

Resource management and live migration

2006 — Process Containers

Developed by Google, Linux Kernel feature that limits, accounts for, and isolates the resource usage of a collection of processes

2007 — Merger of Control Groups in to Linux Kernel

Merged in to Linux Kernel and renamed to control groups

2008 — Aix workload partition

Sofware implementation of OS level virtualization that provides app environment isolation and resource control

LXC Linux Containers

Provided a userspace interface for Linux kernel containment features

Namespaces

Linux kernel that isolates and virtualizes system resources of a collection of processes

2011 — Cloud Foundry Warden

The open Platform as a Service project

2013 — Lmctfy

Open source version of Google's container stack, which provides Linux application containers

Docker

Software container management platform that automates deployment of applications

2014 — Rocket

CoreOS's container management platform

Kubernetes

Automating deployment, scaling, and management of containerized applications

2016 — Apache Mesos Marathon

Production-grade container orchestration platform

Windows Container

Windows server containers and Hyper-V containers

Container Ecosystem Evoluti

Rocket

CoreOS's container management platform

undry

Kubernetes

Platform as a oject

Automating deployment, scaling, and management of containeri-zed applications

05 2006

2008 2C

I 2013 2014 2016

Process Containers

Developed by Google, Linux Kernel feature that limits, accounts for, and isolates the resource usage of a col-lection of processes

LXC Linux Containers

Provided a userspace interfa-ce for Linux kernel contain-ment features

Namespaces

Linux kernel that isolates and virtualizes system resources of a collection of processes

Lmctfy

Open source version of Google's container stack, which provides Linux application containers

Apache Mesos Marathon

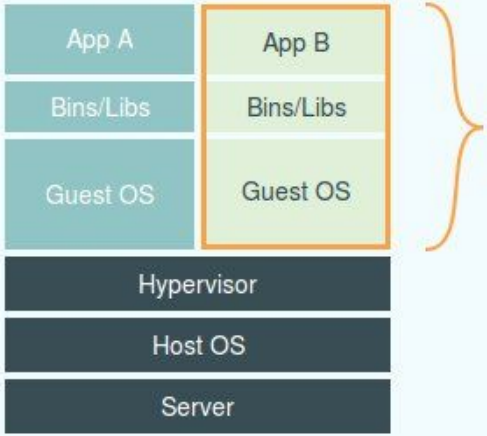
Production-grade container orchestration platform

Docker

Software container manage-ment platform that automates deployment of applications

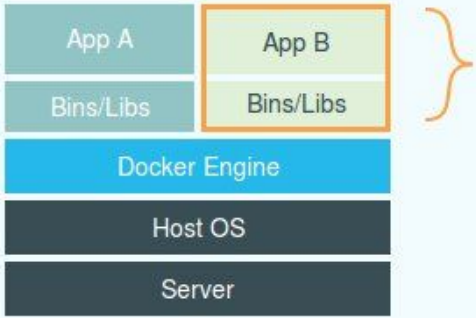
Windows Container

Windows server containers and Hyper-V containers



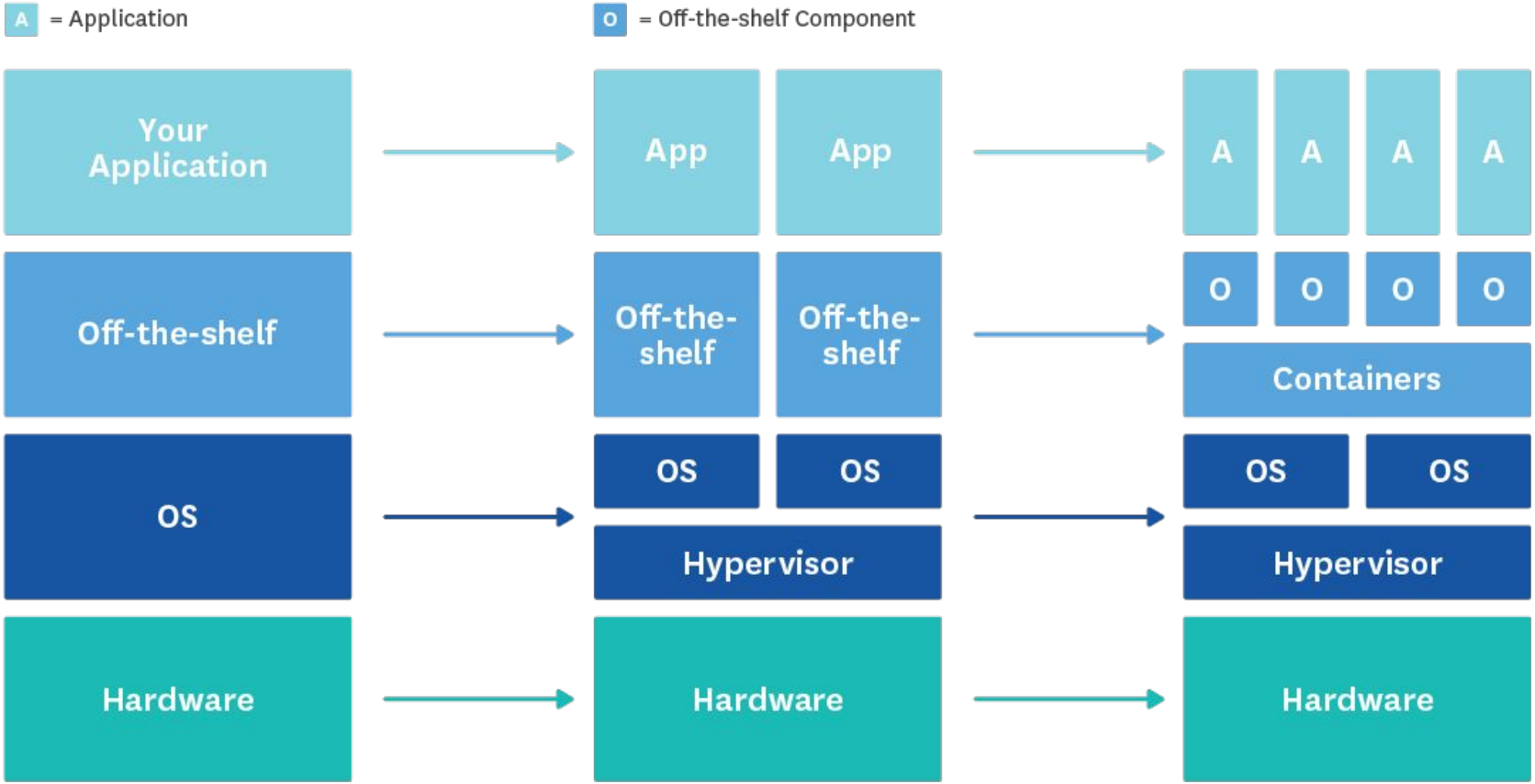
Virtual Machines

Each virtualized application includes not only the application - which may be only 10s of MB - and the necessary binaries and libraries, but also an entire guest operating system - which may weigh 10s of GB.



Docker

The Docker Engine container comprises just the application and its dependencies. It runs as an isolated process in userspace on the host operating system, sharing the kernel with other containers. Thus, it enjoys the resource isolation and allocation benefits of VMs but is much more portable and efficient.


A = Application
O = Off-the-shelf Component
Your Application
Off-the-shelf
OS
Hardware
App
App
Off-the-shelf
Off-the-shelf
OS
OS
Hypervisor
Hardware
A
A
A
A
O
O
O
O
Containers
OS
OS
Hypervisor
Hardware

Workshop: creating and running a docker container

soaked SABJA SEEDS - 2 tbsp

cooked FALOODA SEV - 2 tbsp

MANGO PIECES - 2 tbsp

ROSE SYRUP - 1 tsp

MANGO JELLY (optional) - 2 tbsp

MANGO PULP - 1/4th cup

chilled MILK - 3 tbsp

MANGO ICE CREAM - 1 scoop

TUTTI FRUTTI - few

filename: Dockerfile

```
FROM ubuntu:latest

MAINTAINER me@email.com v0.1

RUN apt-get update \
        && apt-get install -y nginx

COPY nginx.conf /etc/nginx/nginx.conf


EXPOSE 80


CMD ["nginx", "-g", "daemon off;"]
```

Use the latest version of ubuntu as the base image.

You could add this note on who created/maintains the image.

Run these commands in the image. E.g. software installs.

Add this local file(s)/dir to the image. E.g. data, config files.

Allow port 80 to be accessed when the image is run.

Execute this command.

Trying out Docker

Navigate to:

https://katacoda.com/courses/docker/playground

```
mkdir d1 && cd d1
```

```
curl -LJO
https://raw.githubusercontent.com/sathishvj/kubernetes101/master/d1/Dockerfile
```

```
docker images
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
redis	latest	b6dddb991dfa	2 months ago	107MB
ubuntu	latest	2d696327ab2e	2 months ago	122MB
alpine	latest	76da55c8019d	2 months ago	3.96MB

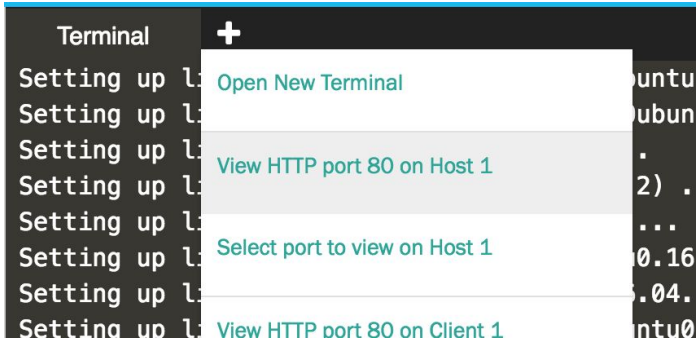
`docker build . -t mynginx`

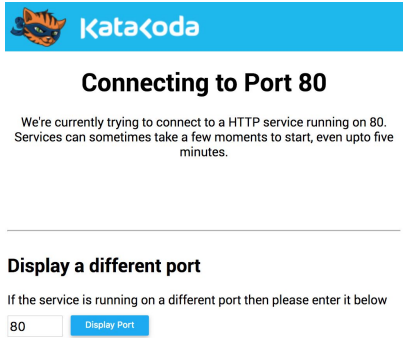
- Takes *Dockerfile* as default from current directory
- Names the image *mynginx*

```
Sending build context to Docker daemon  2.048kB
Step 1/4 : FROM ubuntu:latest
 ---> 2d696327ab2e
Step 2/4 : RUN apt-get update          && apt-get install -y nginx
 ---> Running in ba0e26c0df41
Get:1 http://archive.ubuntu.com/ubuntu xenial InRelease [247 kB]
Get:2 http://archive.ubuntu.com/ubuntu xenial-updates InRelease [102 kB]
Get:3 http://archive.ubuntu.com/ubuntu xenial-backports InRelease [102 kB]
```

`docker images`

```
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
mynginx             latest              4f71d71e2eec        7 seconds ago       218MB
redis               latest              b6dddb991dfa        2 months ago        107MB
ubuntu              latest              2d696327ab2e        2 months ago        122MB
alpine              latest              76da55c8019d        2 months ago        3.96MB
```



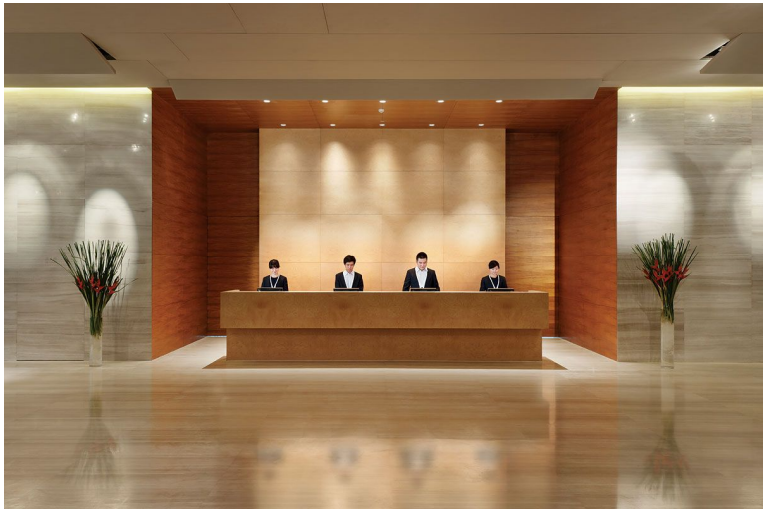



```
docker run -p 80:80 mynginx
```

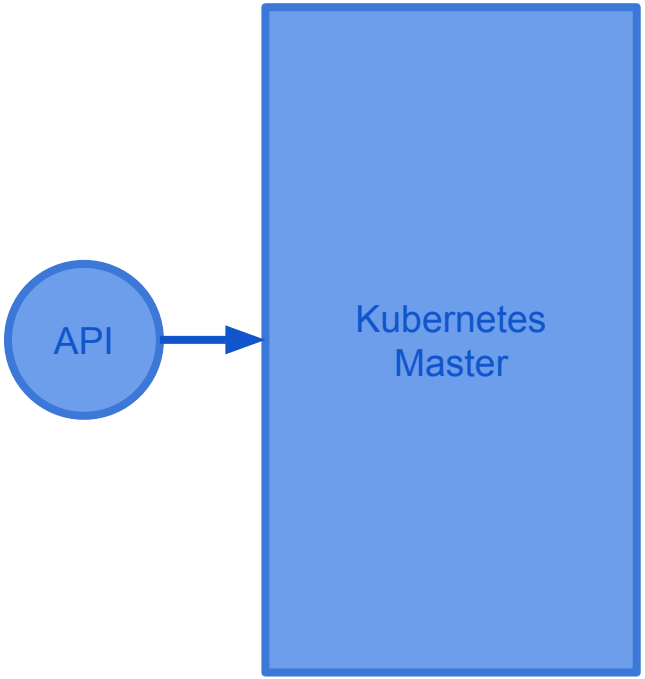
kubernetes

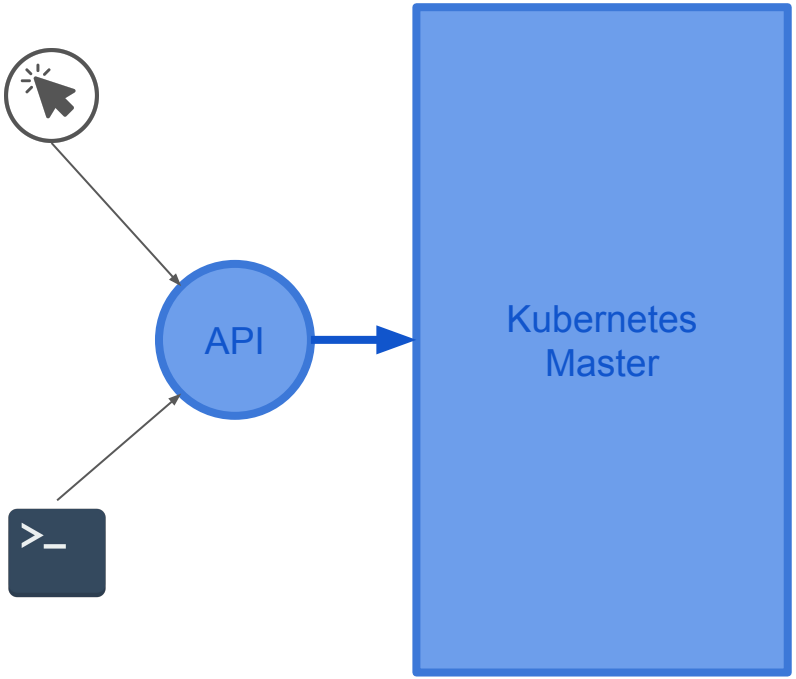


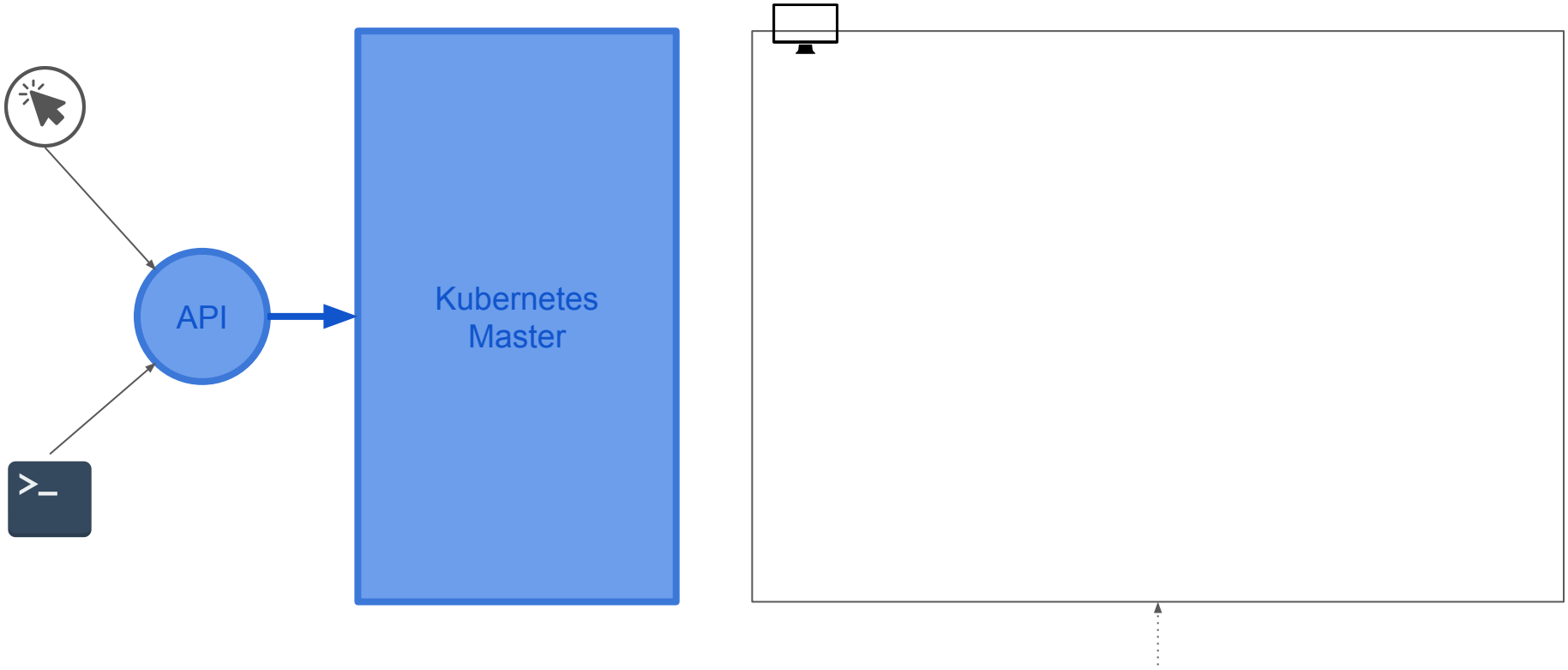


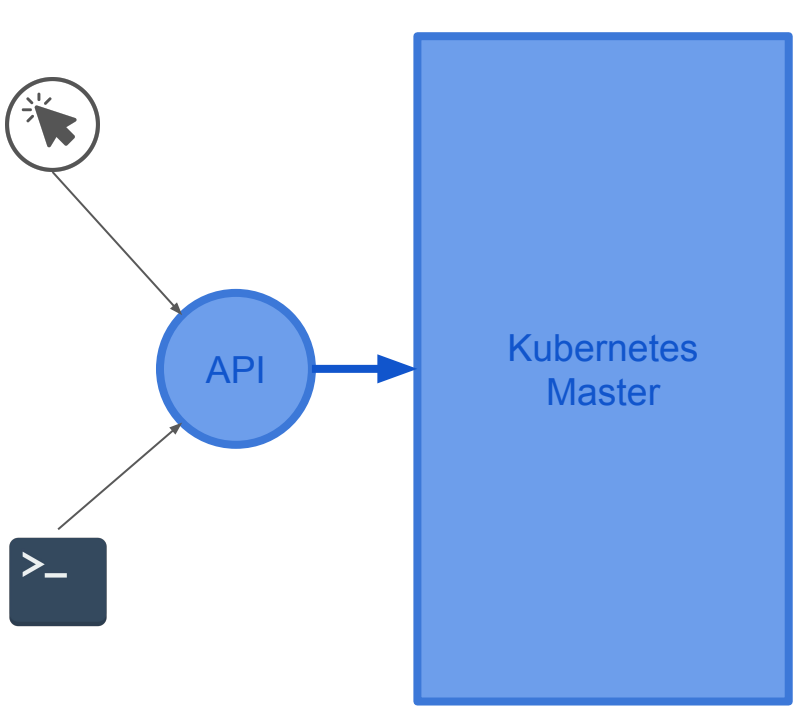



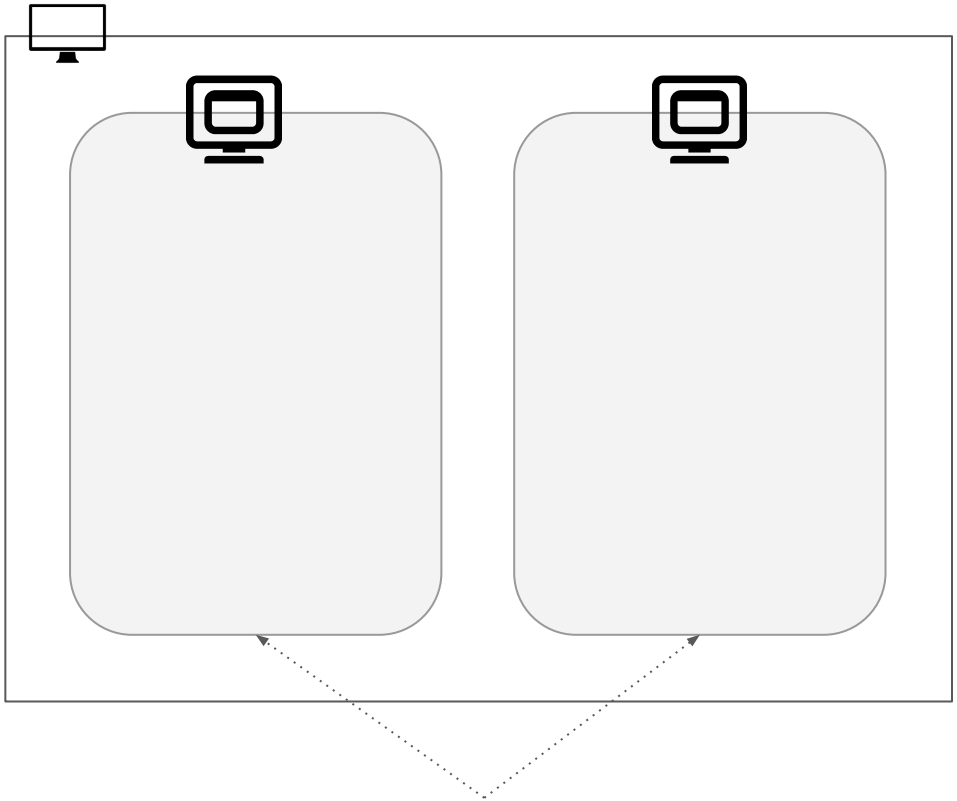

Kubernetes
Master


API
Kubernetes
Master

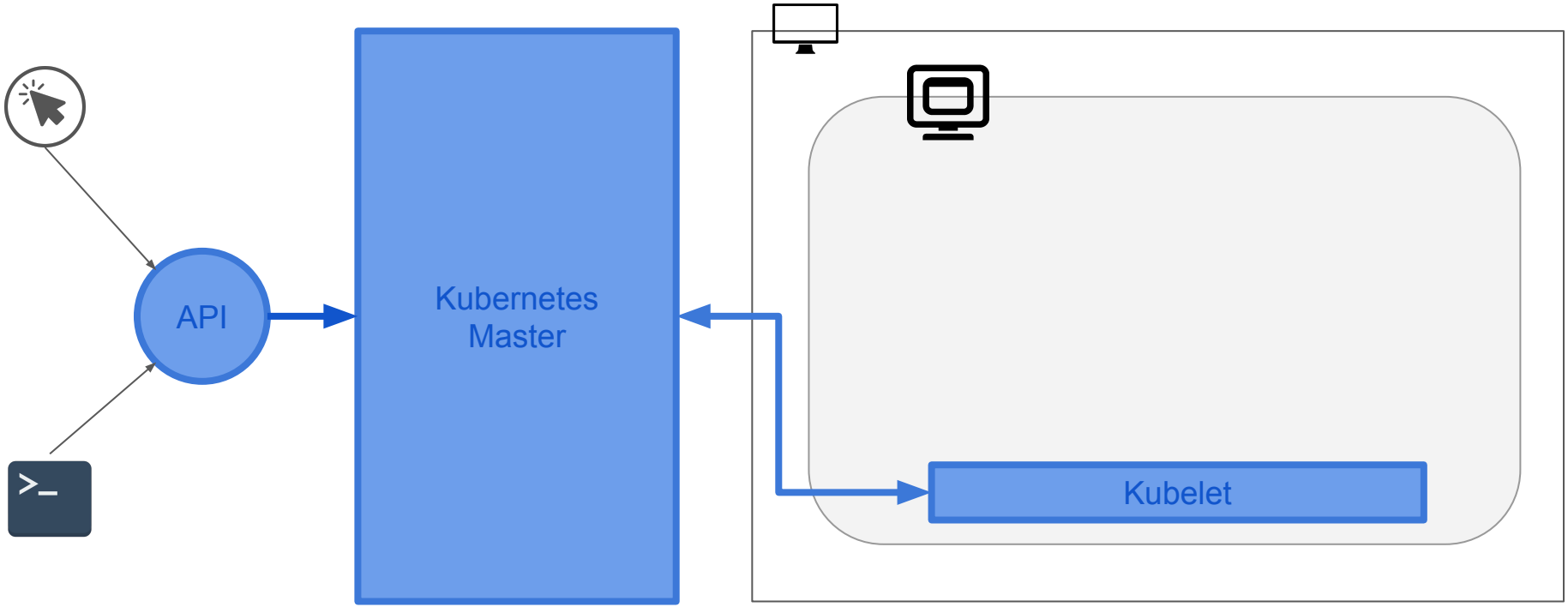

API
Kubernetes
Master
>_

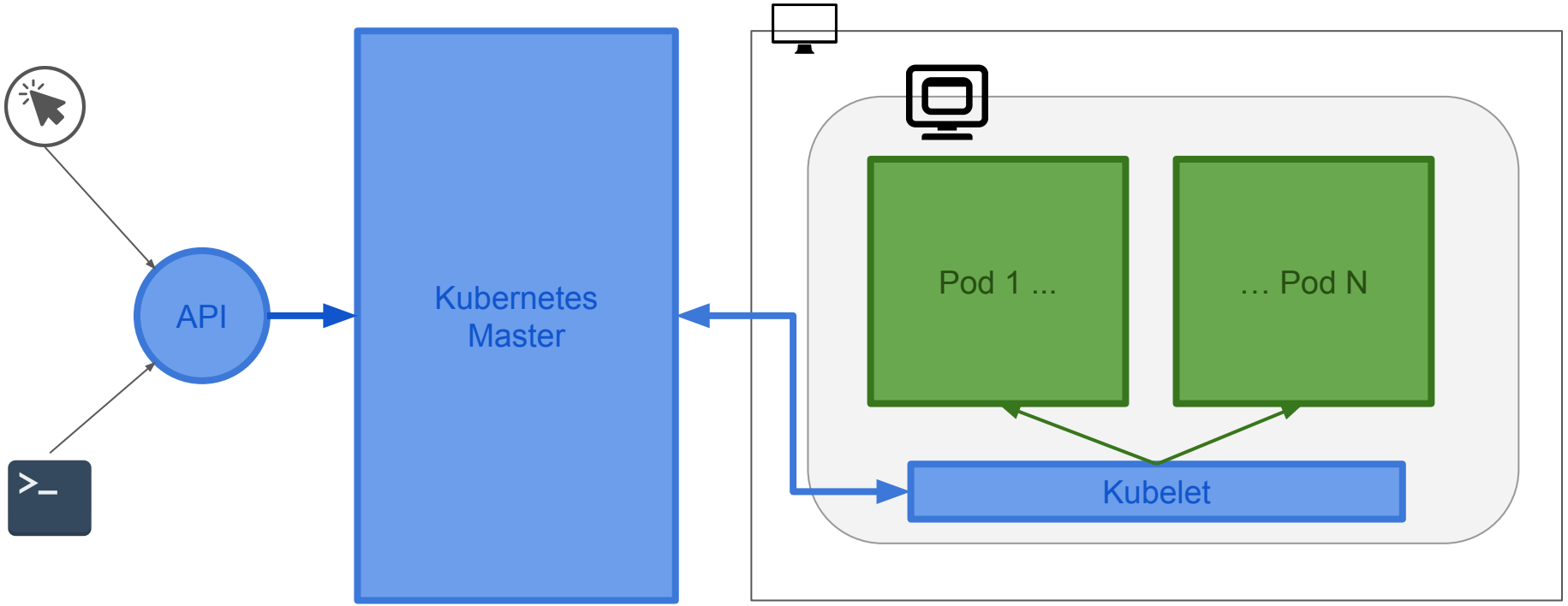

API
Kubernetes
Master
Computer Hardware +
Main OS
This could be a node, but usually not.

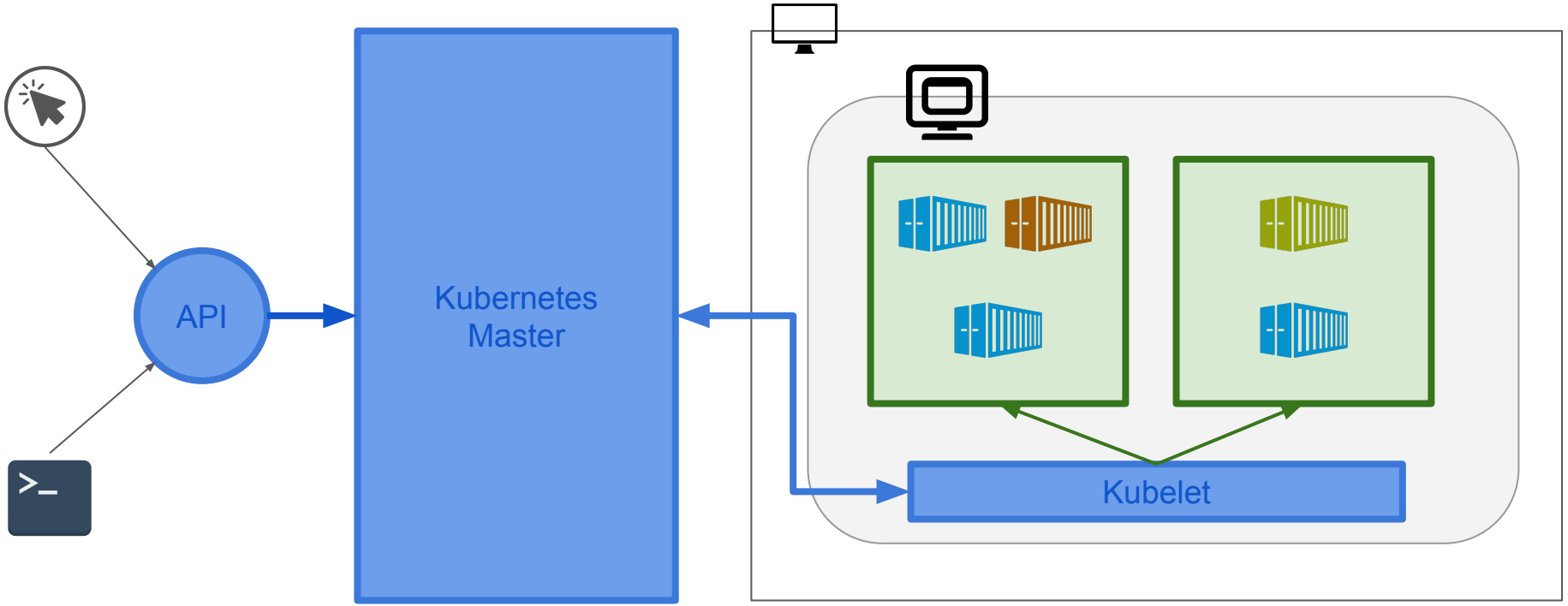


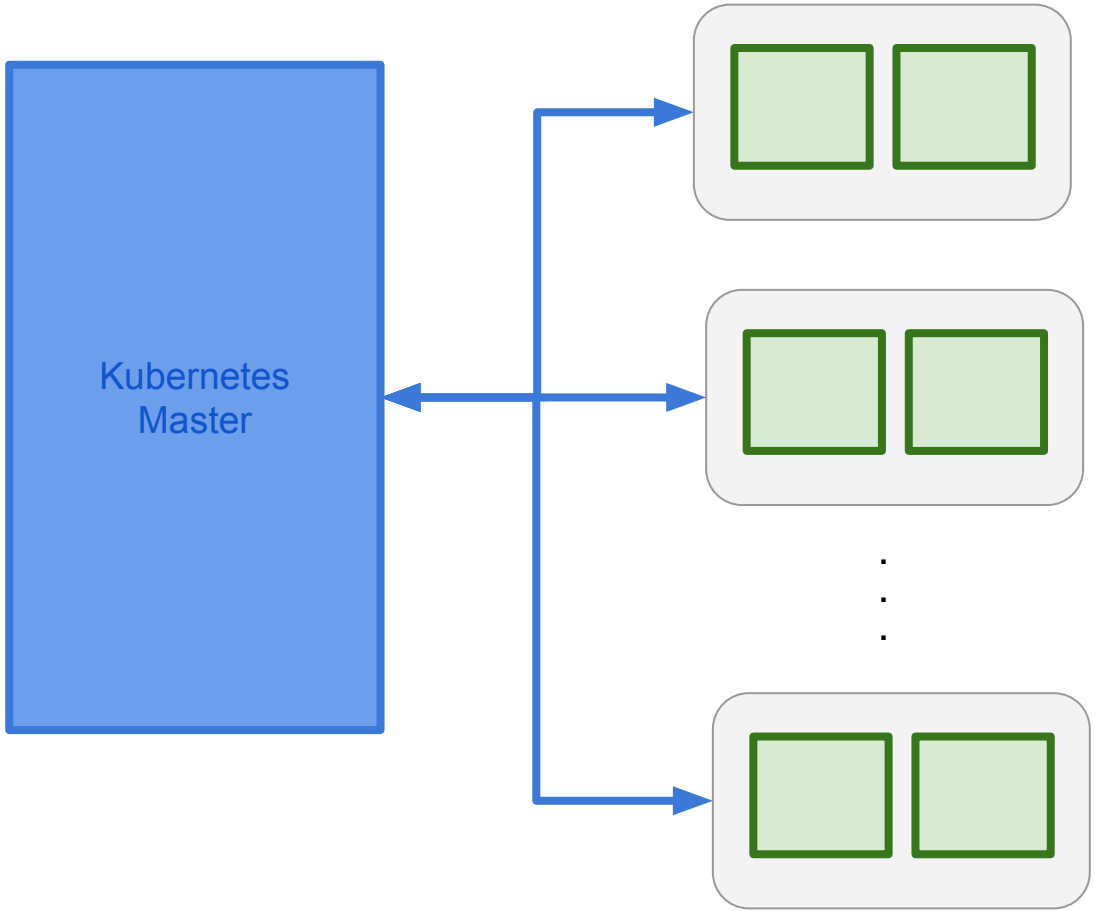


Virtual Machine: Node

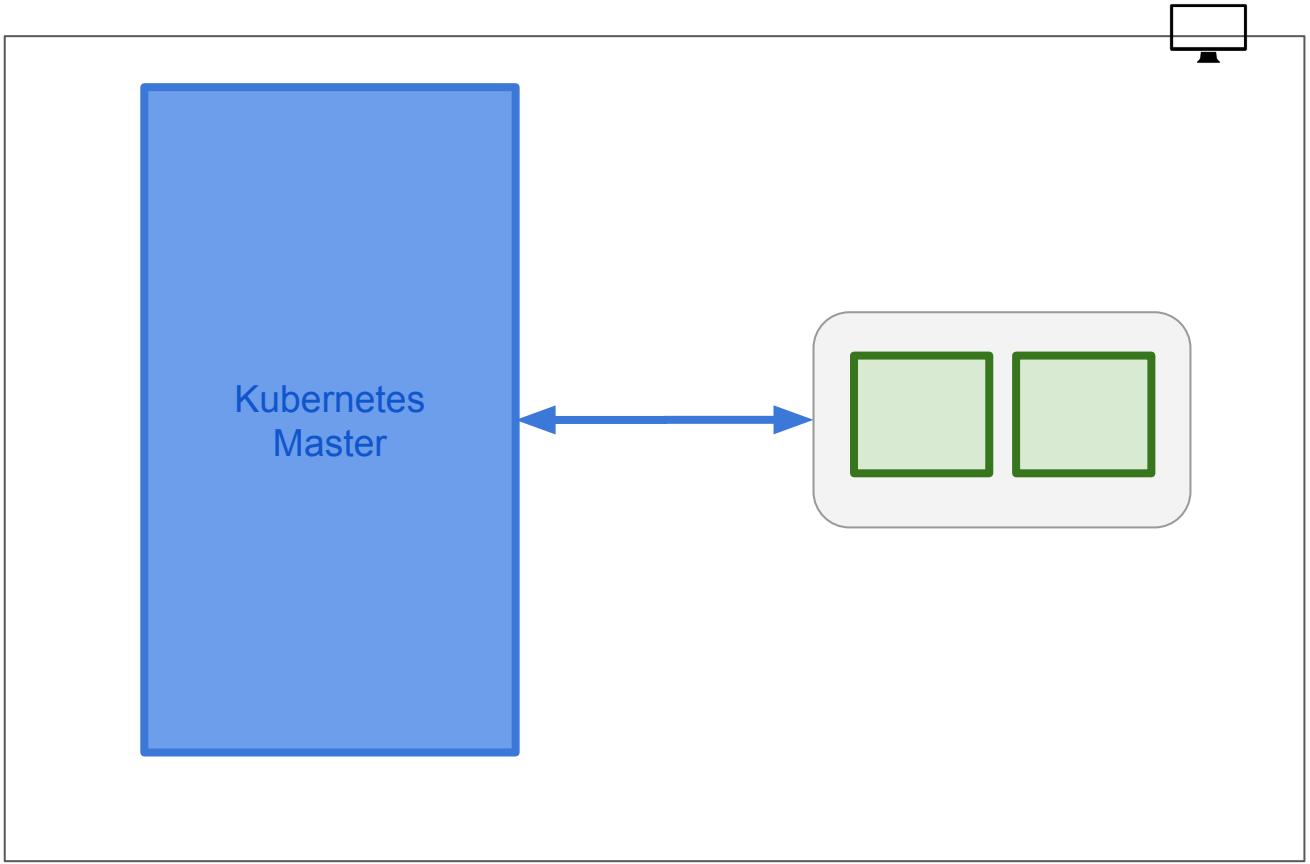

API
Kubernetes
Master
Kubelet

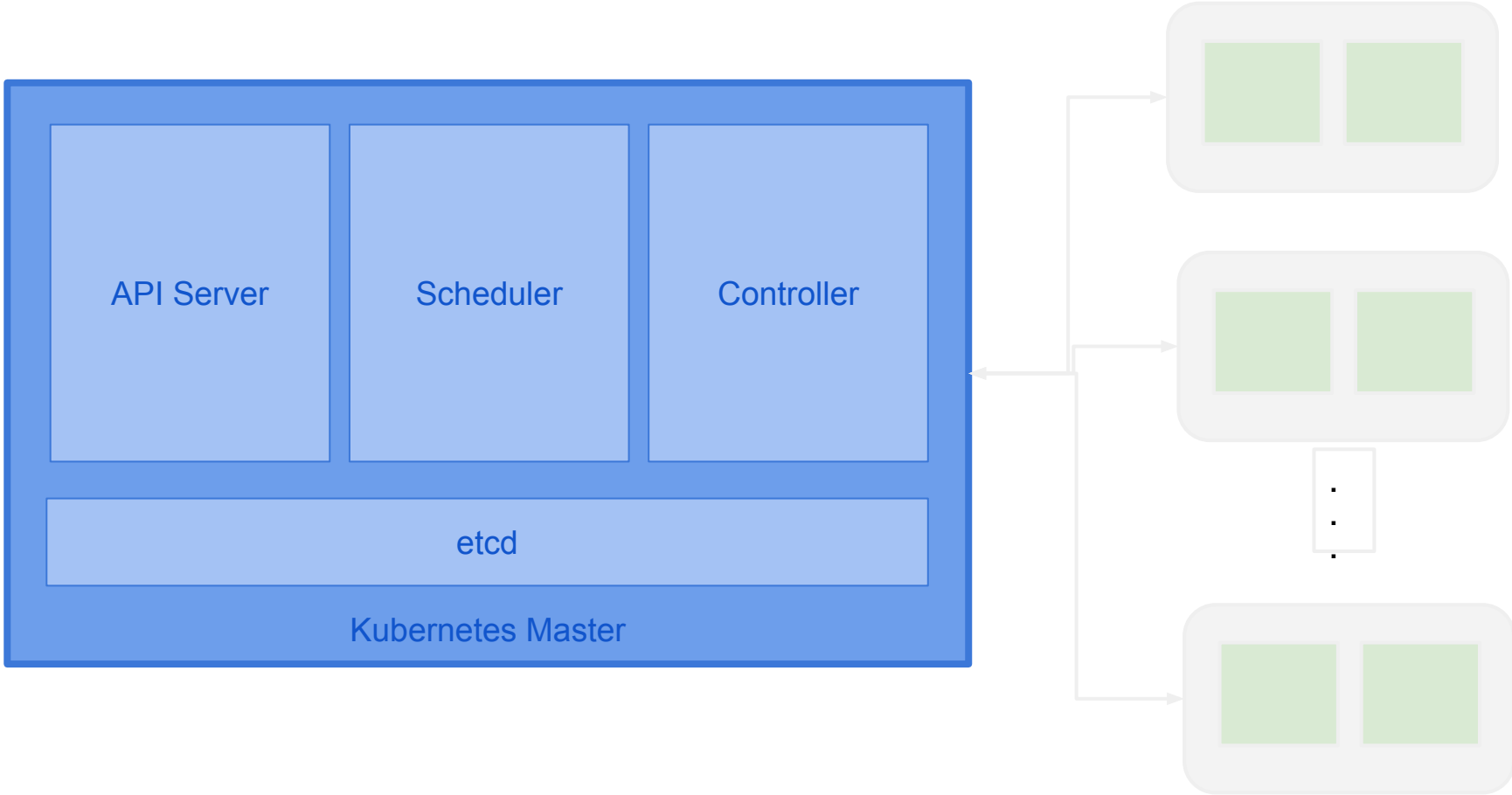

API
Kubernetes
Master
Pod 1 ...
… Pod N
Kubelet


API
Kubernetes
Master
Kubelet


Kubernetes
Master

Minikube/Minishift



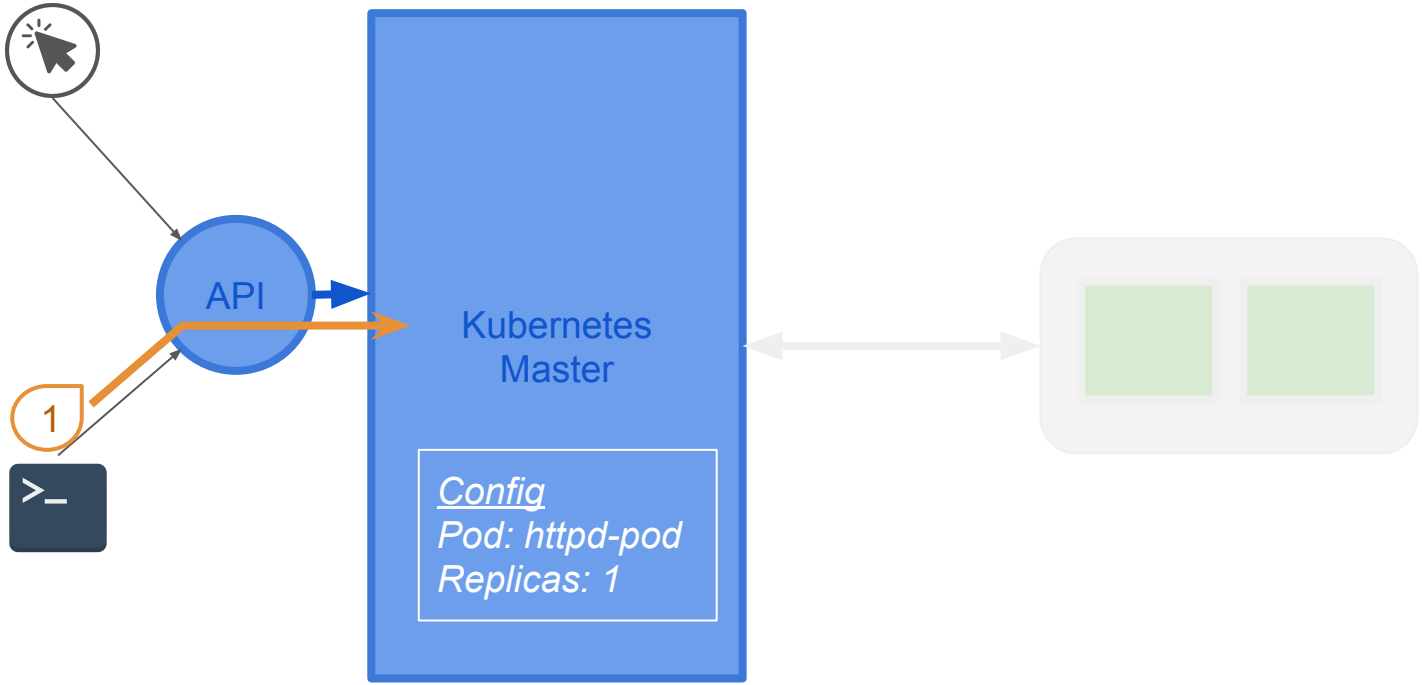

API Server
Scheduler
Controller
etcd
Kubernetes Master

Imperative: “the how”

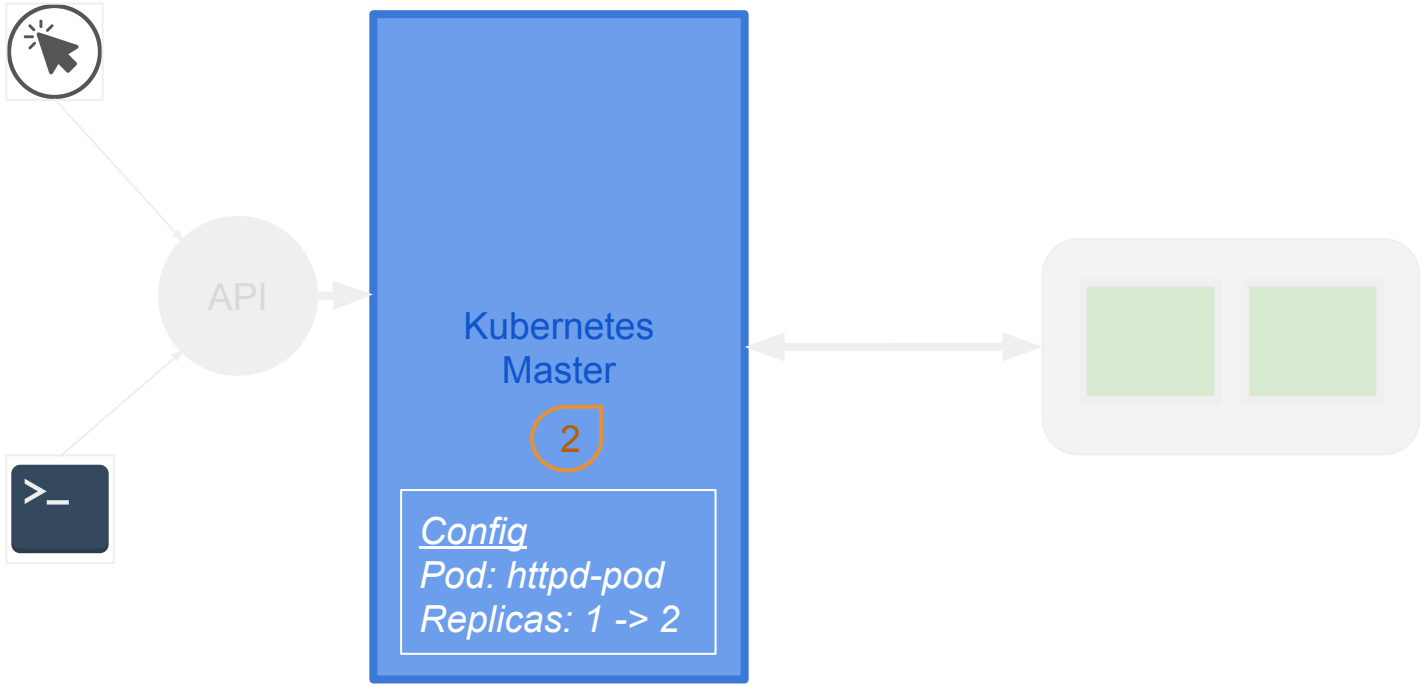
do this, then do this, then do this

Declarative: “the what”

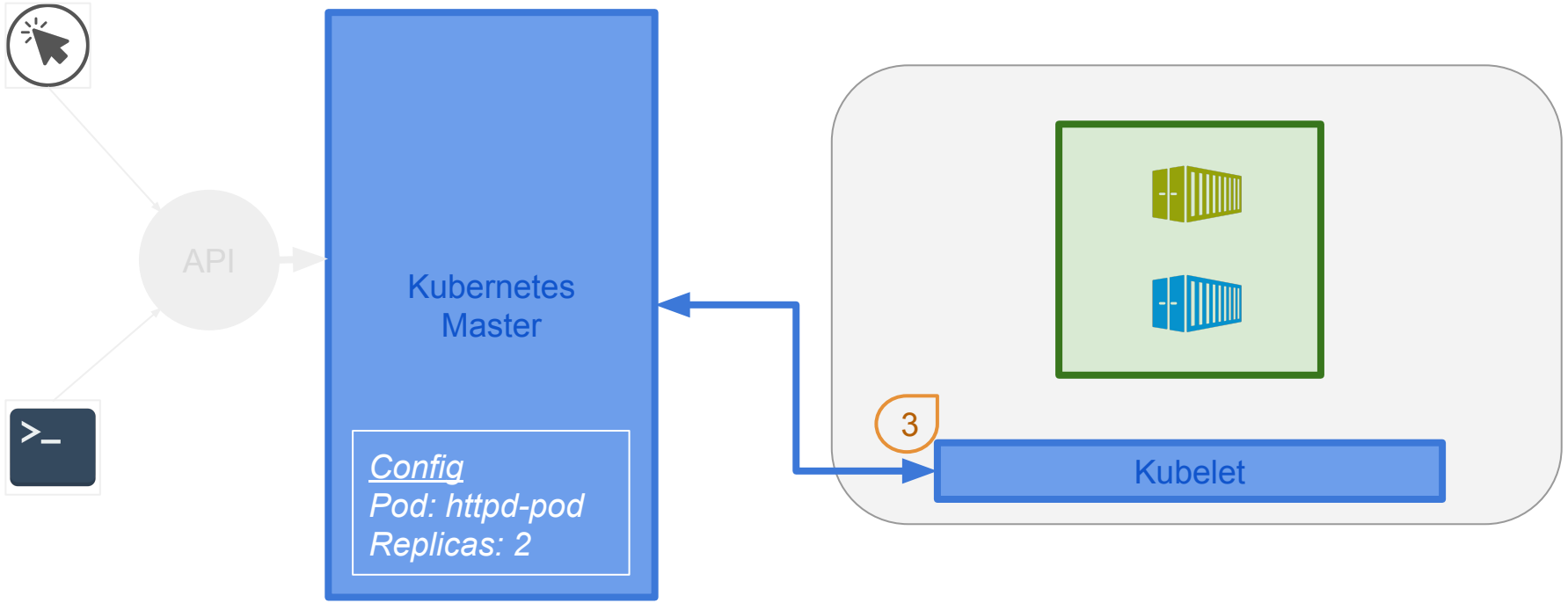
I want this.



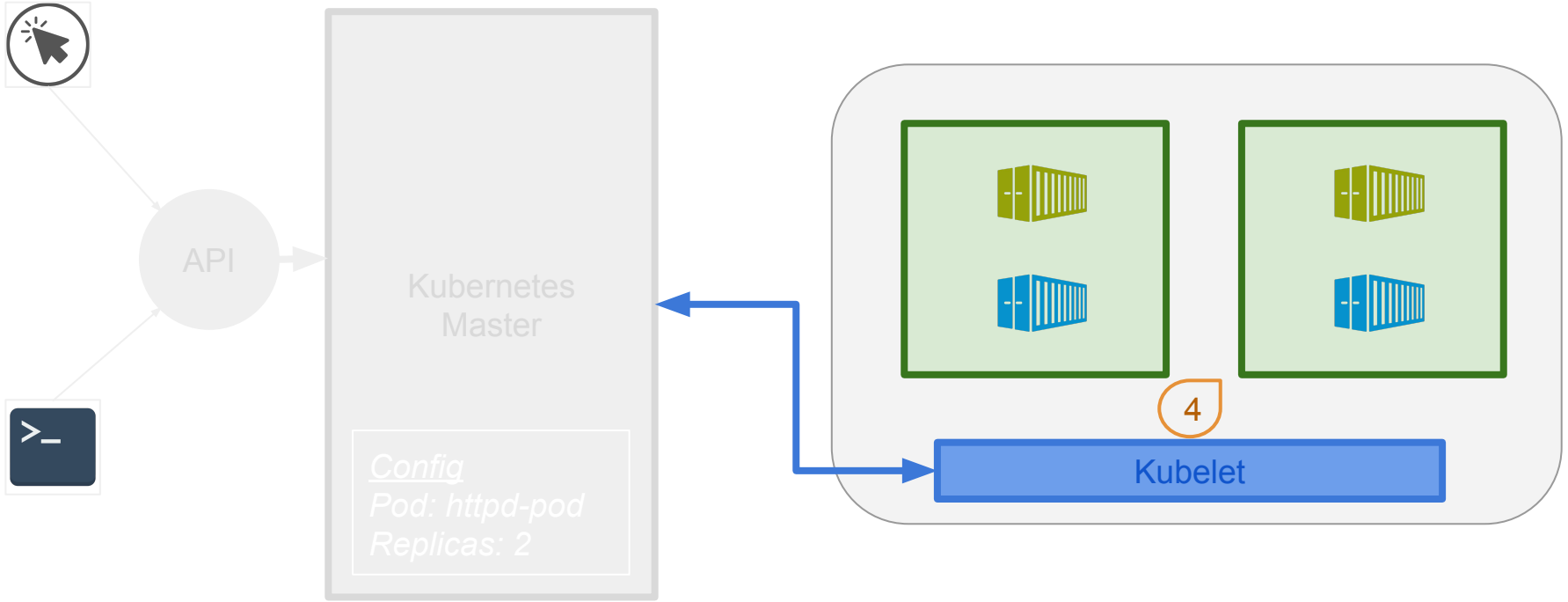
1. External command issued to effect a change in configuration. Ex. Make 2 instances of httpd-pod



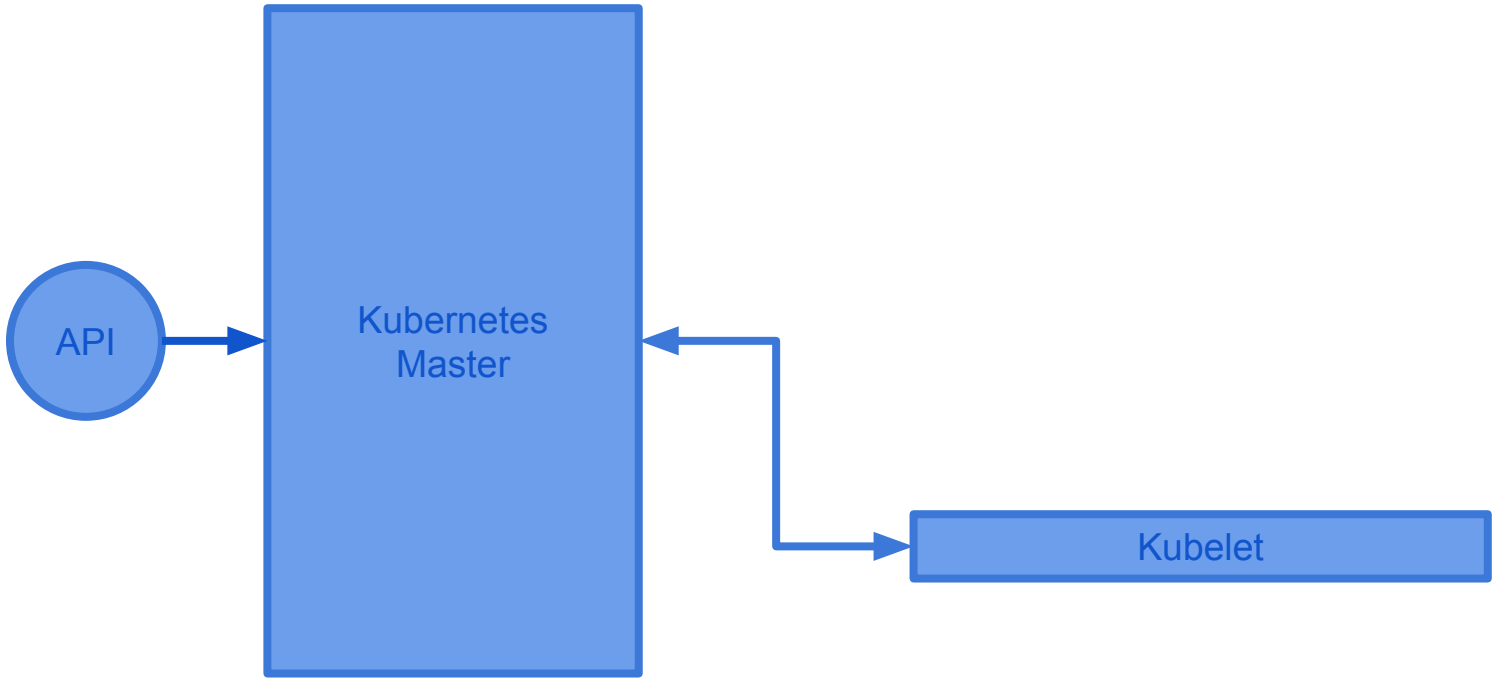
2. Master updates its internal configuration.



3. Kubelets poll and read configuration. Sees mismatch in system ‘truth’.



4. Kubelets updates pods to match.



A cluster with a single node and single master ready to accept api calls.

Workshop: kubernetes

Starting and using Kubernetes

- Step 1: Preferably create a new VM on your computer
  - Download and install virtualbox
  - Install an OS like Fedora/Ubuntu on virtualbox
  - Start the VM instance
  - Install kubernetes tools as below …
- Step 2: create a kubernetes cluster
  - kubeadm: multi node (or)
  - minikube: single node - *we'll use this* (or)
  - minishift: single node version of Red Hat OpenShift
- Step 3:
  - kubectl: use command line interface to control the cluster

Trying out Kubernetes

Navigate to:

https://katacoda.com/courses/kubernetes/launch-single-node-cluster

Also referencing: http://kubernetesbyexample.com/

Start a Cluster

```
which minikube
```

```
/home/scrapbook/.bin/minikube
```

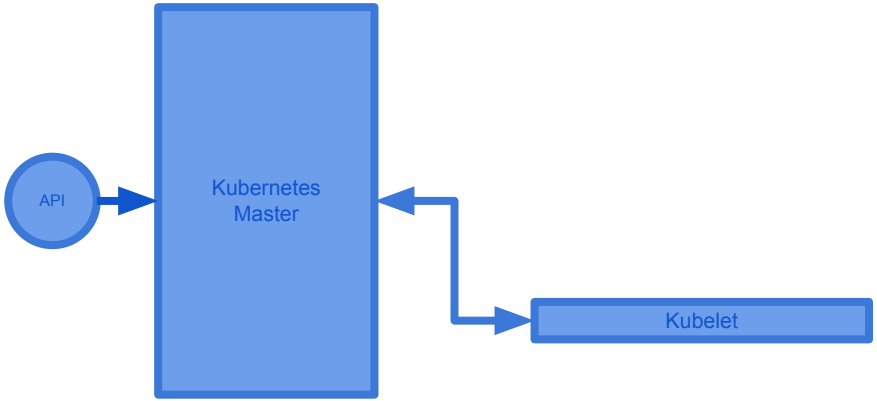
```
minikube version
```

```
minikube version: v0.17.1-katacoda
```

```
minikube start
```

kubectl cluster-info

```
Kubernetes master is running at http://host01:8080
heapster is running at http://host01:8080/api/v1/namespaces/kube-system/services/heapster/proxy
kubernetes-dashboard is running at http://host01:8080/api/v1/namespaces/kube-system/services/kubernetes-dashboard/proxy
monitoring-grafana is running at http://host01:8080/api/v1/namespaces/kube-system/services/monitoring-grafana/proxy
monitoring-influxdb is running at http://host01:8080/api/v1/namespaces/kube-system/services/monitoring-influxdb/proxy
```



Are there any pods running now? Can you construct the command to get a list of running pods?

kubectl get <?>

Run Deployment + Pods

* from image

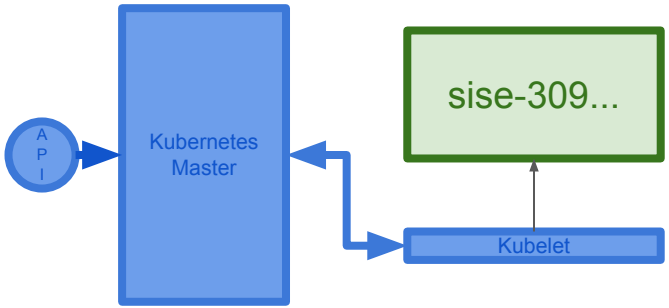
```
kubectl run sise --image=mhausenblas/simpleservice:0.5.0
```

- Through this workshop we'll try to construct commands instead of trying to remember or byheart them.
  - So, get an idea of the general structure of commands.
- Main command immediately follows *kubectl*
- *run* requires a name for the cluster and is mandatory.
- Options follow "--".

```
deployment "sise" created
```

```
? get a list of pods now
```

```
NAME                    READY     STATUS    RESTARTS   AGE
sise-3098977012-205gx   1/1       Running   0          2m
```



- But what's running inside the pod?

Navigate to: https://hub.docker.com/r/mhausenblas/simpleservice/~/dockerfile/

PUBLIC | AUTOMATED BUILD

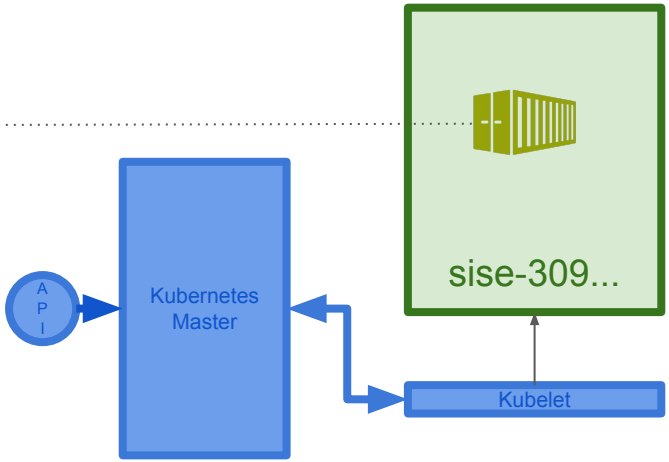
mhausenblas/simpleservice ☆

Last pushed: 7 months ago

Repo Info | Tags | Dockerfile | Build Details

Dockerfile

```
FROM python:2.7-onbuild
MAINTAINER Michael Hausenblas
ENV REFRESHED_AT 2017-04-24T13:50
CMD [ "python", "./simpleservice.py" ]
```

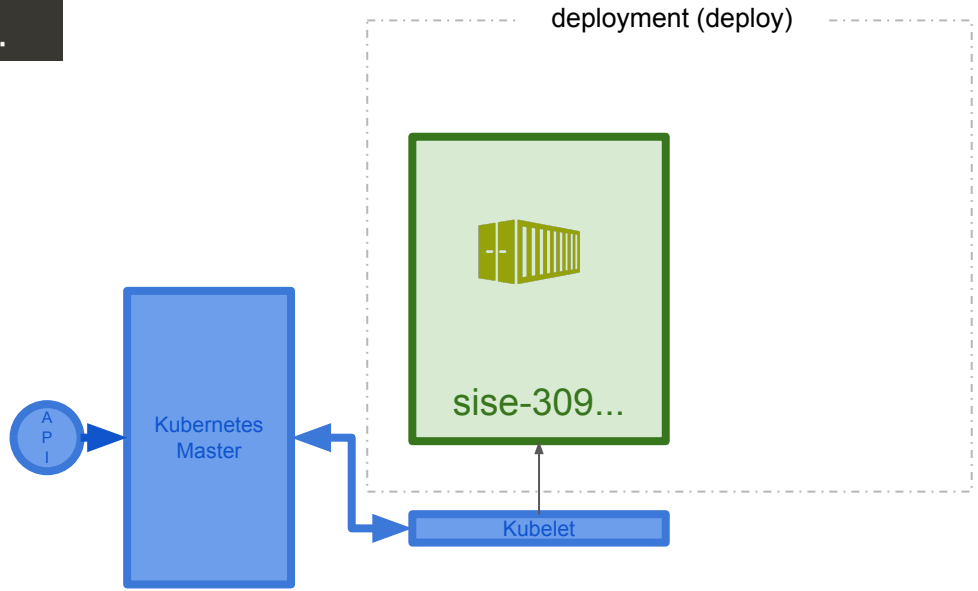


Navigate to: https://github.com/mhausenblas/simpleservice/blob/master/simpleservice.py

kubectl help run

```
Create and run a particular image, possibly replicated.

Creates a deployment or job to manage the created container(s).
```



```
kubectl get deploy
```

```
NAME      DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
sise      1         1         1            0           11s
```

```
kubectl get pods
```

```
NAME                    READY     STATUS              RESTARTS   AGE
sise-3098977012-lnrp1   0/1       ContainerCreating   0          15s
```

k8s automatically created pod names.

Delete Deployments and Pods

Can you construct the command to delete the deployment (a.k.a. deploy) ‘sise’?

```
kubectl delete deploy sise
```

```
deployment "sise" deleted
```

Is cluster still running? *Hint: cluster-info*.

```
kubectl delete deploy sise
```

```
deployment "sise" deleted
```

Is cluster still running? *Hint: cluster-info*.

Redeploy the earlier one again.

```
kubectl run sise --image=mhausenblas/simpleservice:0.5.0
```

Can you construct the command to delete the pod?

Once you delete the pod, check pods again. What do you see?

kubectl delete pod sise-...

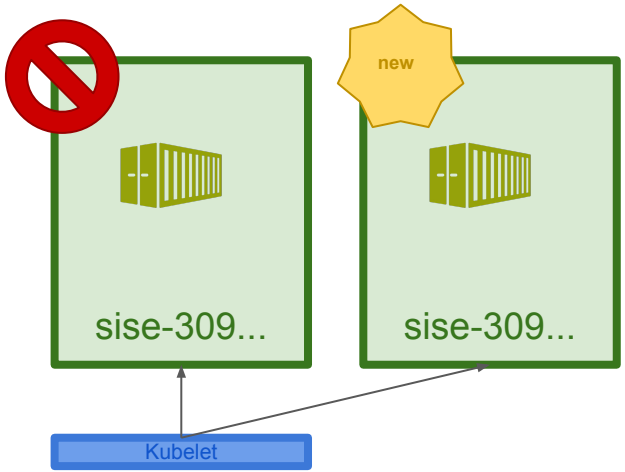
```
pod "sise-3098977012-478pn" deleted
```

kubectl get pods

```
NAME                    READY     STATUS        RESTARTS   AGE
sise-3098977012-478pn   1/1       Terminating   0          26s
sise-3098977012-t3tfk   1/1       Running       0          9s
```

k8s automatically maintains required state.

```
NAME                    READY     STATUS    RESTARTS   AGE
sise-3098977012-t3tfk   1/1       Running   0          2m
```



What is a Pod?

- a group of one or more containers
- shared storage/network, and a specification for how to run the containers
- pod's contents are always co-located and co-scheduled, and run in a shared context
- it contains one or more application containers which are relatively tightly coupled
    - in a pre-container world, they would have executed on the same physical or virtual machine
- The shared context of a pod is a set of Linux namespaces, cgroups, and potentially other facets of isolation.
    - Within a pod's context, the individual applications may have further sub-isolations applied.
- Containers within a pod share an IP address and port space, and can find each other via `localhost`
- Applications within a pod also have access to shared volumes, which are defined as part of a pod and are made available to be mounted into each application's filesystem.
- considered to be relatively ephemeral (rather than durable) entities
- pods are created, assigned a unique ID (UID), and scheduled to nodes where they remain until termination (according to restart policy) or deletion
- When something is said to have the same lifetime as a pod, such as a volume, that means that it exists as long as that pod (with that UID) exists. If that pod is deleted for any reason, even if an identical replacement is created, the related thing (e.g. volume) is also destroyed and created anew.

Pod Management

- Pods are a model of the pattern of multiple cooperating processes which form a cohesive unit of service.
- Pods serve as unit of deployment, horizontal scaling, and replication.
- Colocation (co-scheduling), shared fate (e.g. termination), coordinated replication, resource sharing, and dependency management are handled automatically for containers in a pod.

Create Pods from Config Files

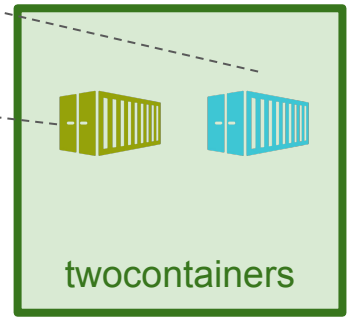
```
apiVersion: v1
kind: Pod
metadata:
  name: twocontainers
spec:
  containers:
  - name: sise
    image: mhausenblas/simpleservice:0.5.0
    ports:
    - containerPort: 9876
  - name: shell
    image: centos:7
    command:
      - "bin/bash"
      - "-c"
      - "sleep 10000"
```

This configuration is only a pod.

The pod will be called ‘twocontainers’

The pod will have 2 containers.

This one is based on the earlier image, will be named ‘sise’, and will expose the port 9876 outside the pod.



```
kubectl create -f
https://raw.githubusercontent.com/mhausenblas/kbe/master/specs/pods/pod.yaml
```

```
kubectl get pods
```

```
NAME                     READY     STATUS    RESTARTS   AGE
sise-3210265840-6p2wd    1/1       Running   0          6m
twocontainers            2/2       Running   0          1m
```

```
kubectl get deploy
```

```
NAME      DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
sise      1         1         1            1           14m
```

No new deployment has been created. Since this is only a Pod config, it used the existing deploy.

Can you construct the command to see logs of a pod?

```
kubectl logs sise-3210265840-6p2wd
```

Logs of this pod default to the only container in it.

```
kubectl logs twocontainers
```

```
Error from server (BadRequest): a container name must be specified for pod twocontainers, choose one of: [sise shell]
```

```
kubectl logs twocontainers -c sise
```

```
2017-12-08T08:38:05 INFO This is simple service in version v0.5.0 listening on port 9876 [at line 142]
2017-12-08T08:41:46 INFO /info serving from localhost:9876 has been invoked from 127.0.0.1 [at line 101]
2017-12-08T08:41:46 INFO 200 GET /info (127.0.0.1) 1.69ms [at line 1946]
```

If there is more than 1, specify the container name.

Can you construct the command to *exec* into the terminal in container ‘shell’?

Hint: *-i -t -- bash*

```
kubectl exec twocontainers -c shell -i -t -- bash
```

```
[root@twocontainers /]#
```

```
curl -s localhost:9876/info
```

```
{"host": "localhost:9876", "version": "0.5.0", "from": "127.0.0.1"}[root@twocontainers /]
```

Containers within this pod are sharing resources like network *localhost*.

Can you construct the command to get the description of a pod?

```
kubectl describe pod twocontainers
```

```
Name:         twocontainers
Namespace:    default
Node:         host01/172.17.0.43
Start Time:   Fri, 08 Dec 2017 20:38:04 +0000
Labels:       <none>
Annotations:  <none>
Status:       Running
IP:           172.18.0.3
Containers:
  sise:
    Container ID:   docker://0738512fe8656b29272958
    Image:          mhausenblas/simpleservice:0.5.0
```

Can you combine them to …

1. exec into the sise-... shell?
2. curl the ip:9876/info of twocontainers' sise container?

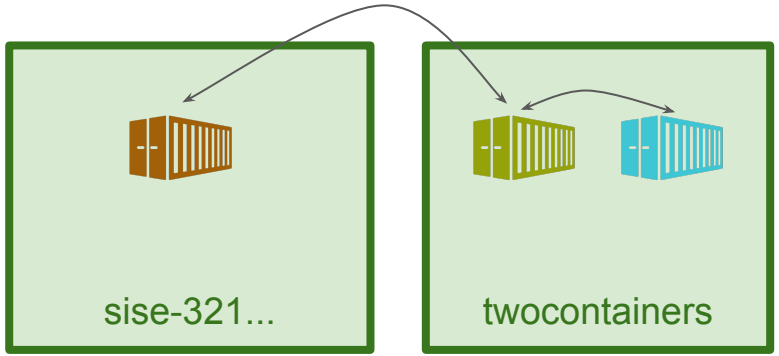
```
kubectl describe pod twocontainers | grep IP
```

```
IP:           172.18.0.3
```

```
kubectl exec sise-3210265840-6p2wd -it -- bash
```

```
# curl 172.18.0.3:9876/info
```

```
{"host": "172.18.0.3:9876", "version": "0.5.0", "from": "172.18.0.2"}
```



Create Pods with Resource Constraints

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: constraintpod
spec:
  containers:
  - name: sise
    image: mhausenblas/simpleservice:0.5.0
    ports:
    - containerPort: 9876
    resources:
      limits:
        memory: "64Mi"
        cpu: "500m"
```

This container will have max limits on memory and cpu.

Use *create -f* to add constraint pod.

Then *describe* it to check.

Create Deployment from a Config file.

```
apiVersion: ~~apps~~ extensions/v1beta1
kind: Deployment
metadata:
  name: sise-deploy
spec:
  replicas: 2
  template:
    metadata:
      labels:
        app: sise
    spec:
      containers:
      - name: sise
        image: mhausenblas/simpleservice:0.5.0
        ports:
        - containerPort: 9876
        env:
        - name: SIMPLE_SERVICE_VERSION
          value: "0.9"
```

This is a deployment.

There should always be 2 pods with these specs.

We will be able to search/filter this pod using this label.

It has only this one container.

```
kubectl create -f
https://raw.githubusercontent.com/sathishvj/kubernetes101/master/k/deployment.yaml
```

```
kubectl get deploy
```

```
NAME          DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
sise-deploy   2         2         2            0           17s
```

```
kubectl get pods
```

```
NAME                           READY     STATUS              RESTARTS   AGE
sise-deploy-3513442901-8h4kj   0/1       ContainerCreating   0          12s
sise-deploy-3513442901-dlxsv   0/1       ContainerCreating   0          12s
```

thank you

@sathishvj