How To Build A Micro-Services Infrastructure in 7 Days

Gil Tayar, October 2017
@giltayar

Wix Internal Hackathon - December 2015

About Me

- My developer experience goes all the way back to the ‘80s.
- Am, was, and always will be a developer
- Currently evangelist and architect at **Applitools**
- Visual Testing

- Sometimes my arms bend back

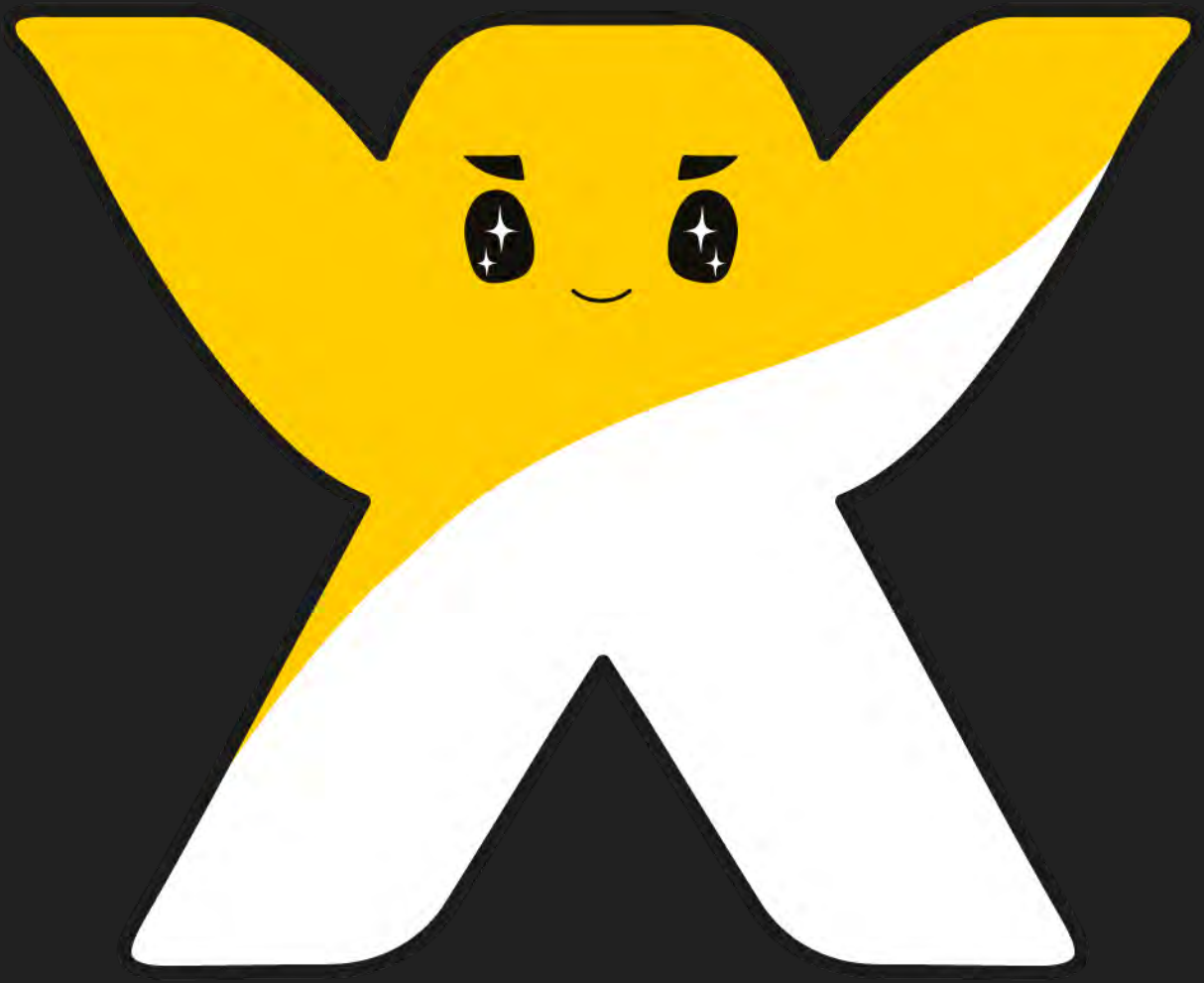
A Movie Script in 6 Parts

1. Prolog
2. The Plan
3. The Job
4. The Change of Plan
5. It All Comes Together
6. Epilog

1. Prol

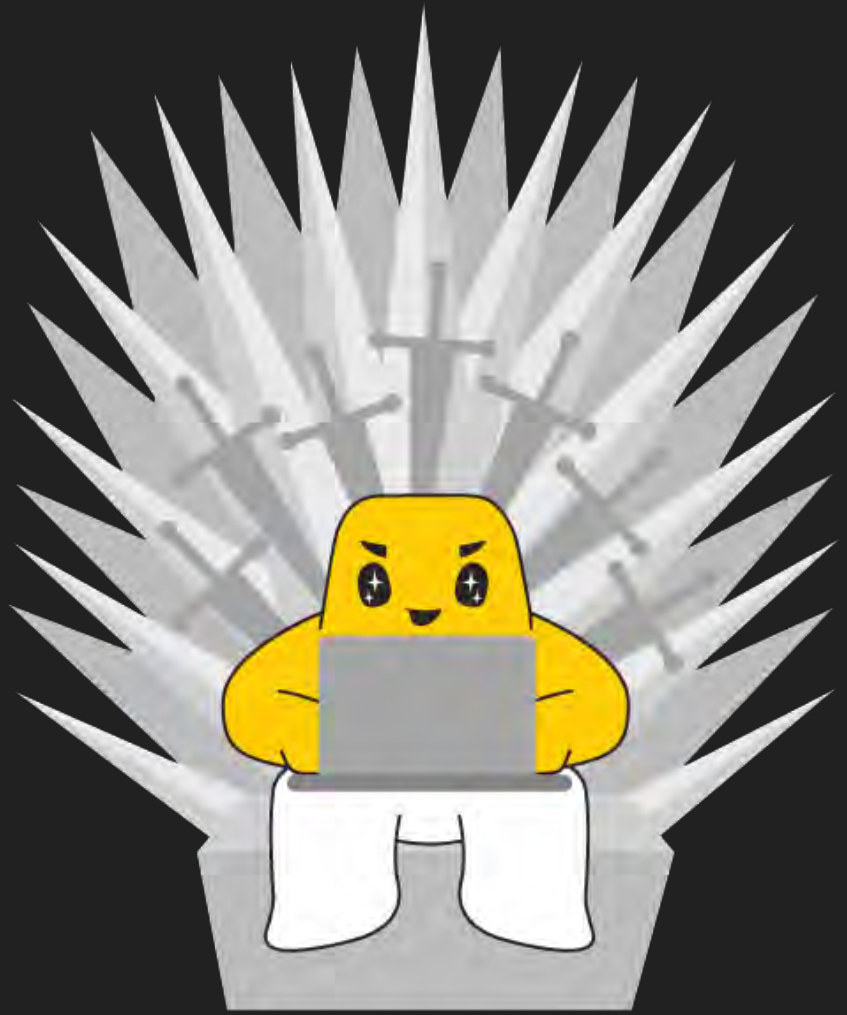
og

Wix



Statistics

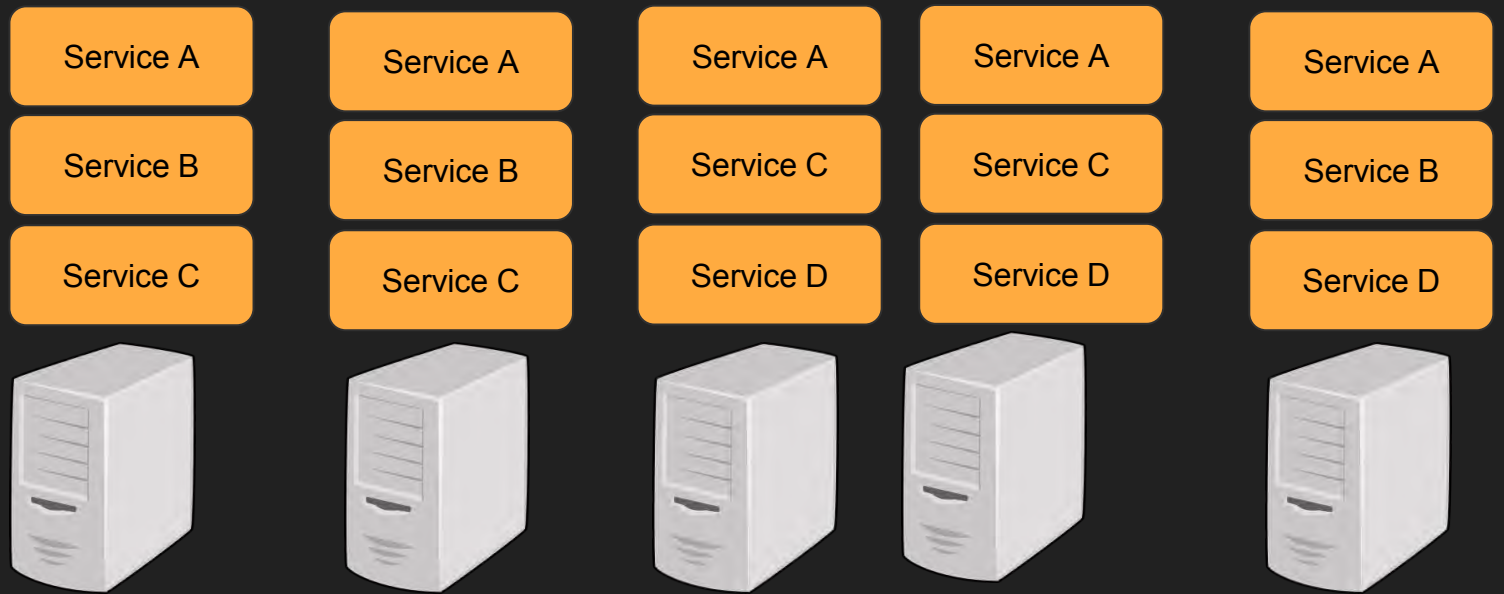
- 100 Million Users
- 3B HTTP Requests/Day
- Over 250 Micro-services
- 150 Deployments/Day



And an Aging Micro-Services Infra!

What is a Micro-services infrastructure?

Deploying
Running
Keeping Healthy

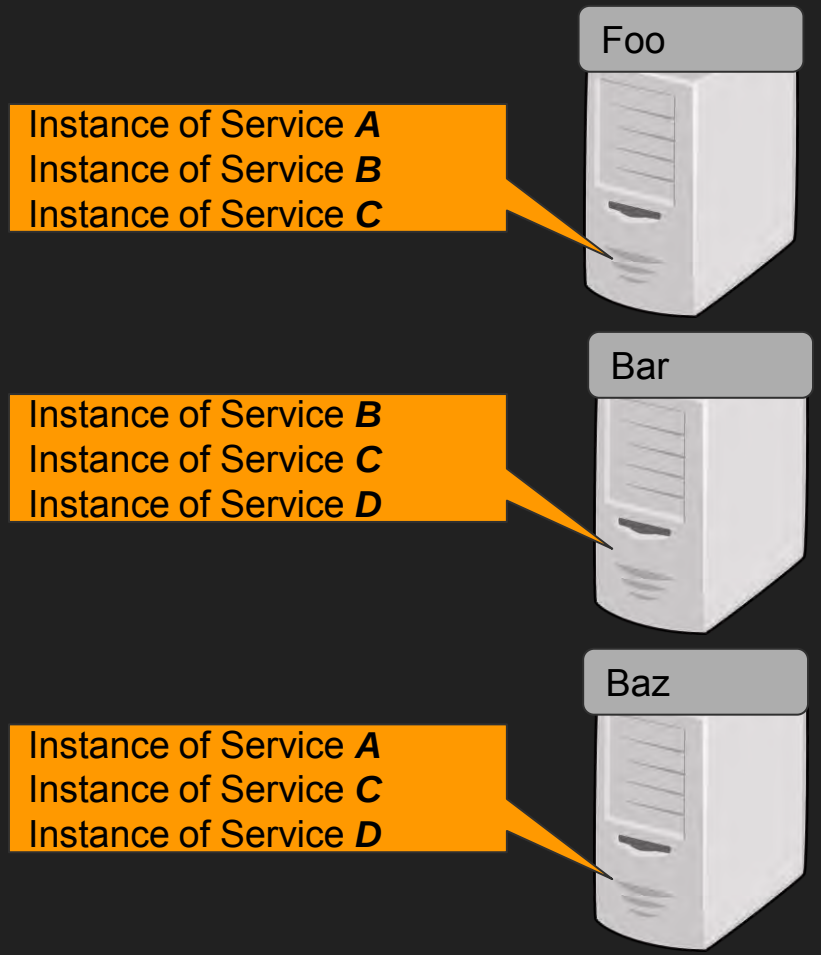


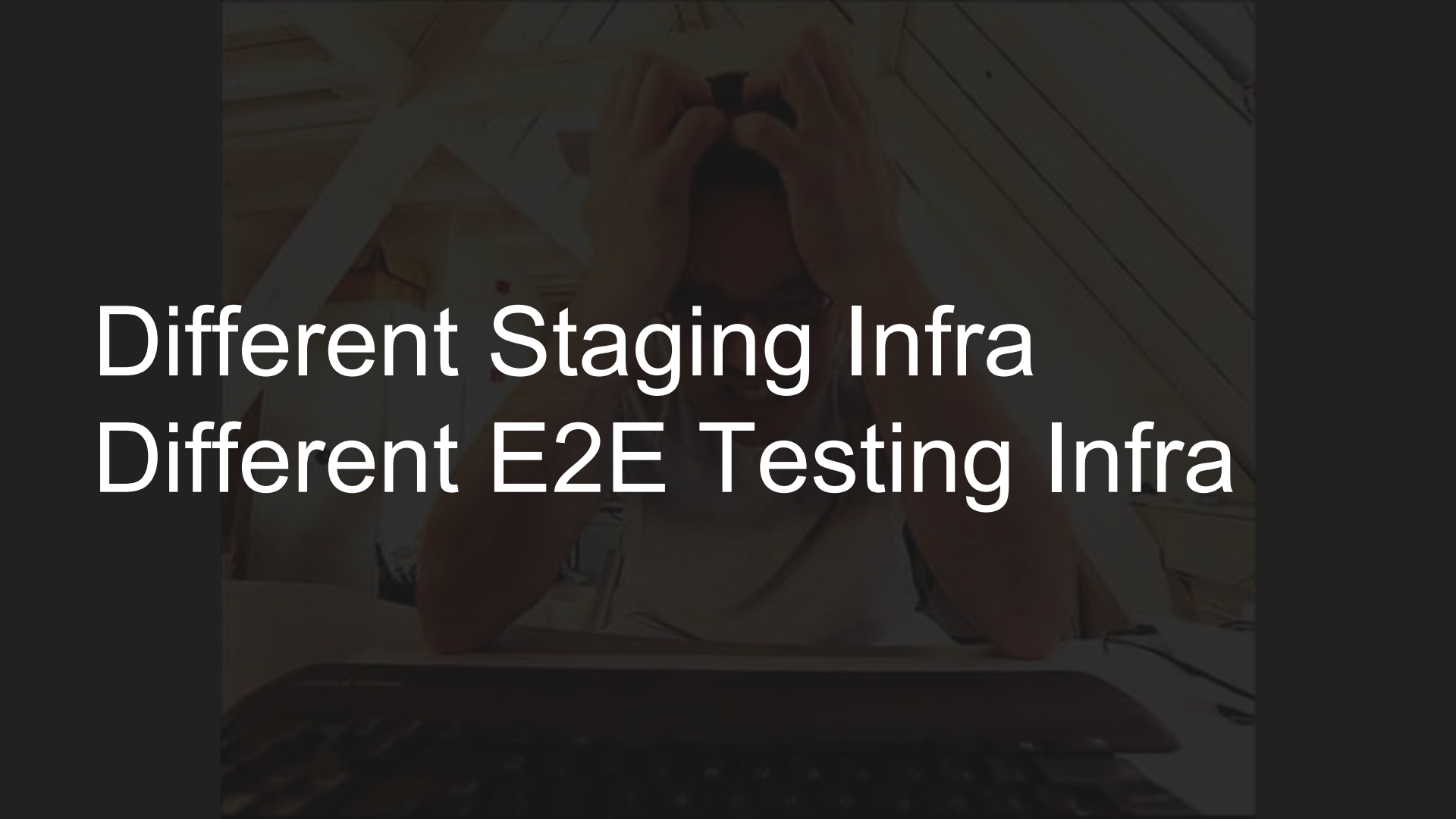
Is It That Bad?

Slow Deployments…

- Chef-based
- Suboptimal Blue/Green Deployment
- Slow Rollbacks

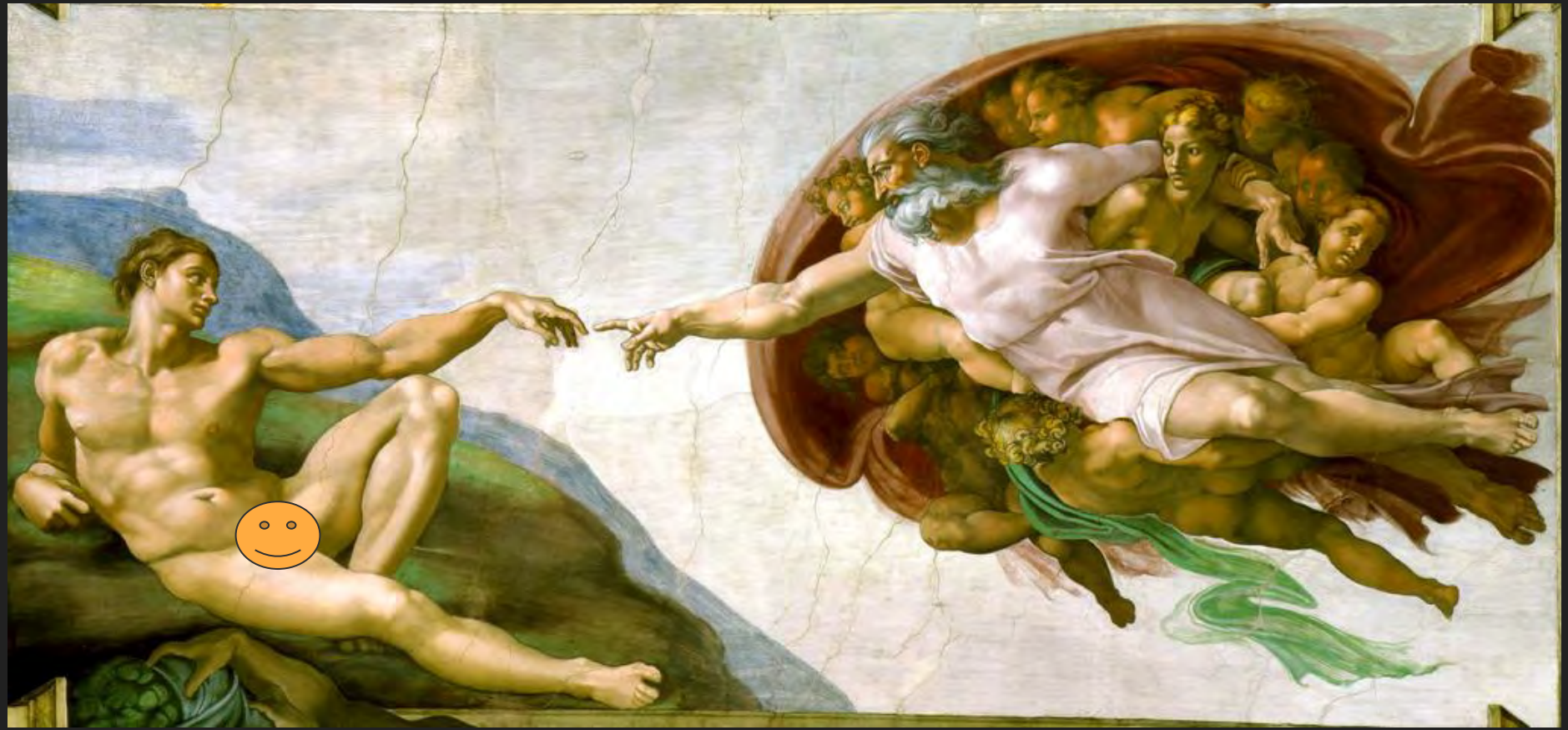
Static Definitions





Different Staging Infra
Different E2E Testing Infra


Can we rebuild it?
In a week?



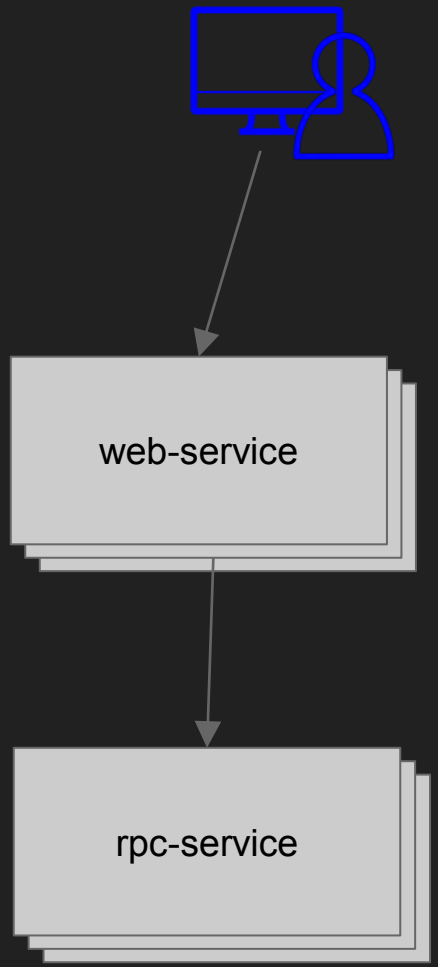

outnow.ch

2. The Plan`

December 15, 2015

The Scope

The Scope (I)



The Scope (II)

```
$ npm install -g apollo

$ apollo artifact-deploy web-service:1.5.0 -n 20

$ curl web-service.demo-domain.com/....

$ apollo artifact-deploy web-service:1.6.0 -n 30
```

The Scope (II)

```
$ npm install -g apollo

$ apollo artifact-deploy web-service:1.5.0 -n 20

$ curl web-service.demo-domain.com/....

$ apollo artifact-deploy web-service:1.6.0 -n 30
```

The Scope (II)

```
$ npm install -g apollo

$ apollo artifact-deploy web-service:1.5.0 -n 20

$ curl web-service.demo-domain.com/....

$ apollo artifact-deploy web-service:1.6.0 -n 30
```

The Scope (II)

```
$ npm install -g apollo

$ apollo artifact-deploy web-service:1.5.0 -n 20

$ curl web-service.demo-domain.com/....

$ apollo artifact-deploy web-service:1.6.0 -n 30
```

The Scope (II)

```
$ npm install -g apollo

$ apollo artifact-deploy web-service:1.5.0 -n 20

$ curl web-service.demo-domain.com/....

$ apollo artifact-deploy web-service:1.6.0 -n 30
```

And we should have some marketing!

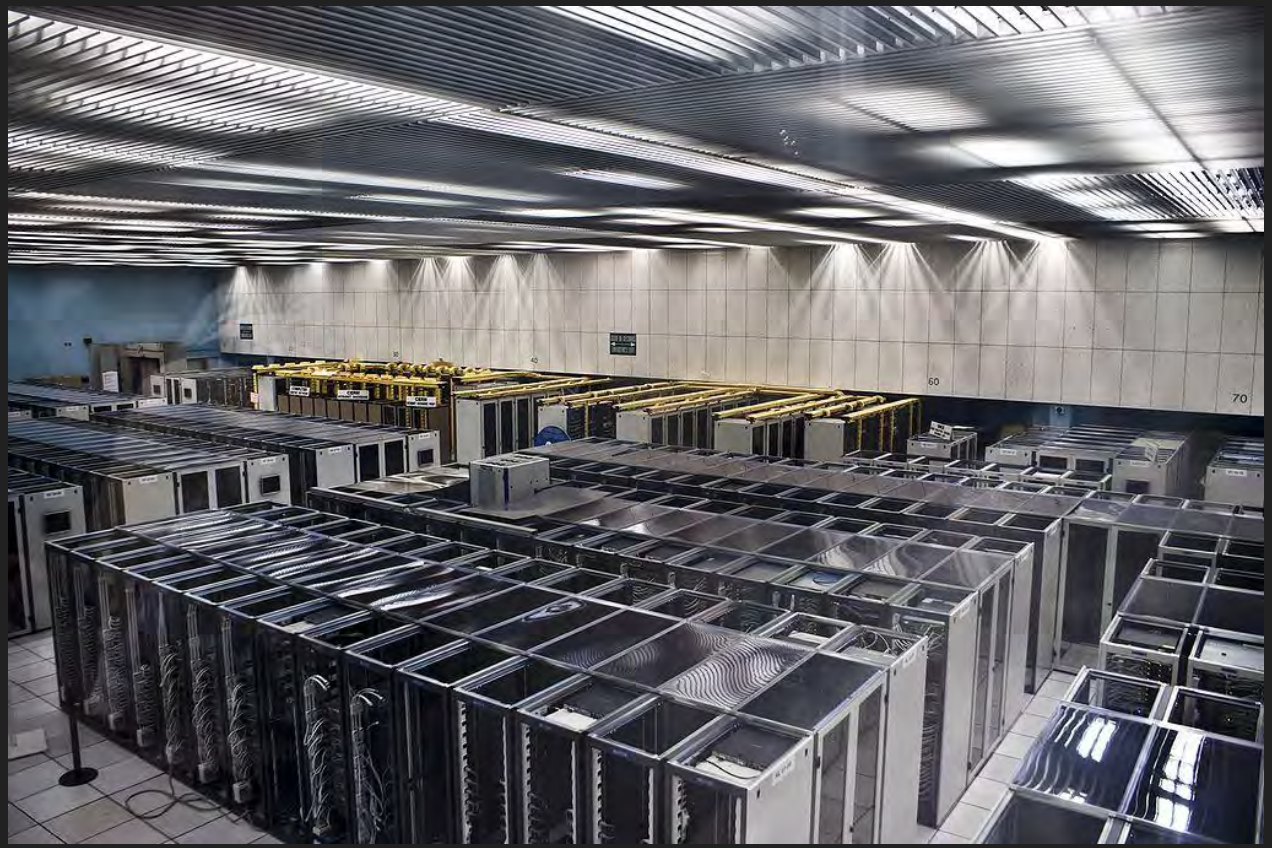
Apollo: Tomorrow's Infrastructure Today

Deploy Like a Boss
'Cause it's Built on Mesos

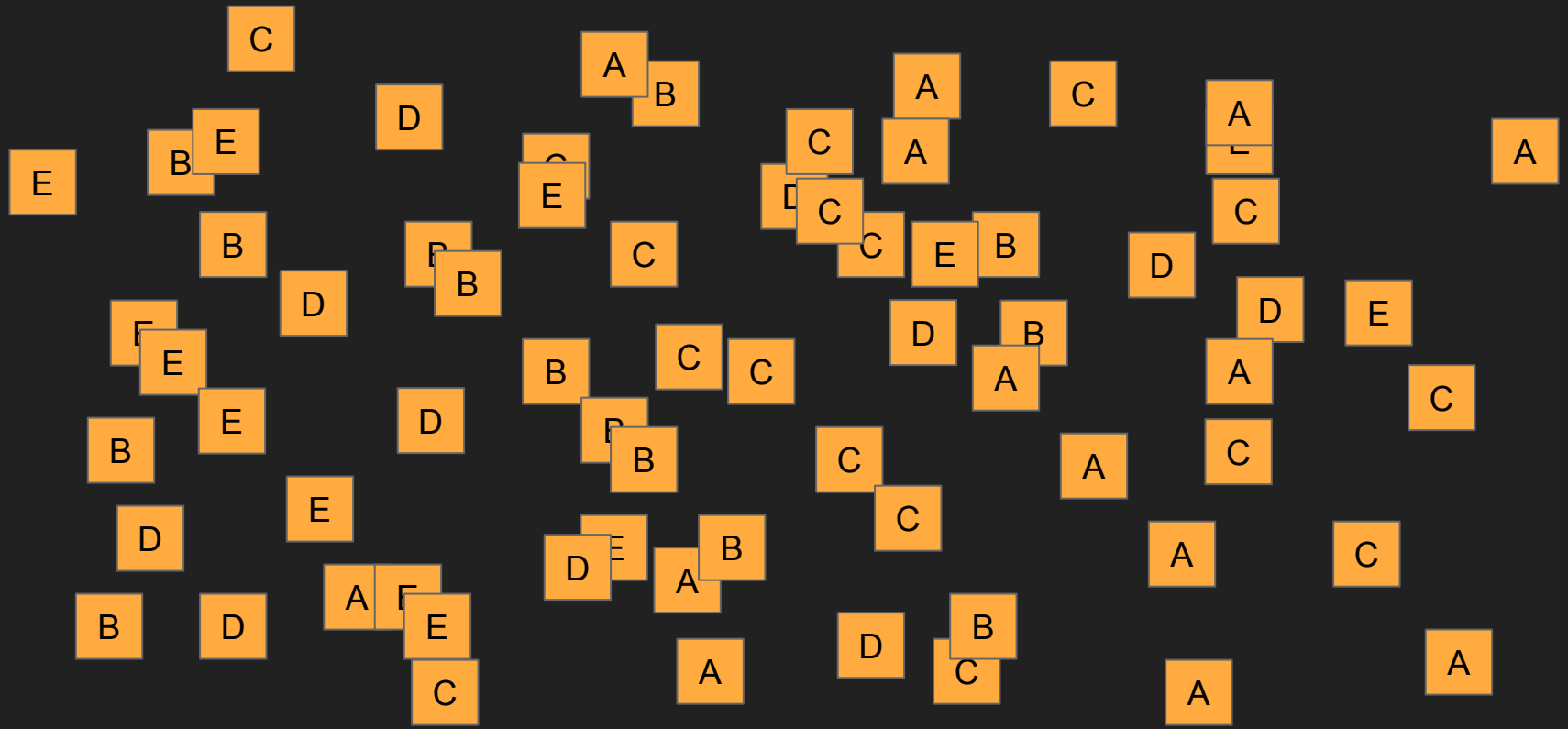
WIXEngineering

The Scheduler

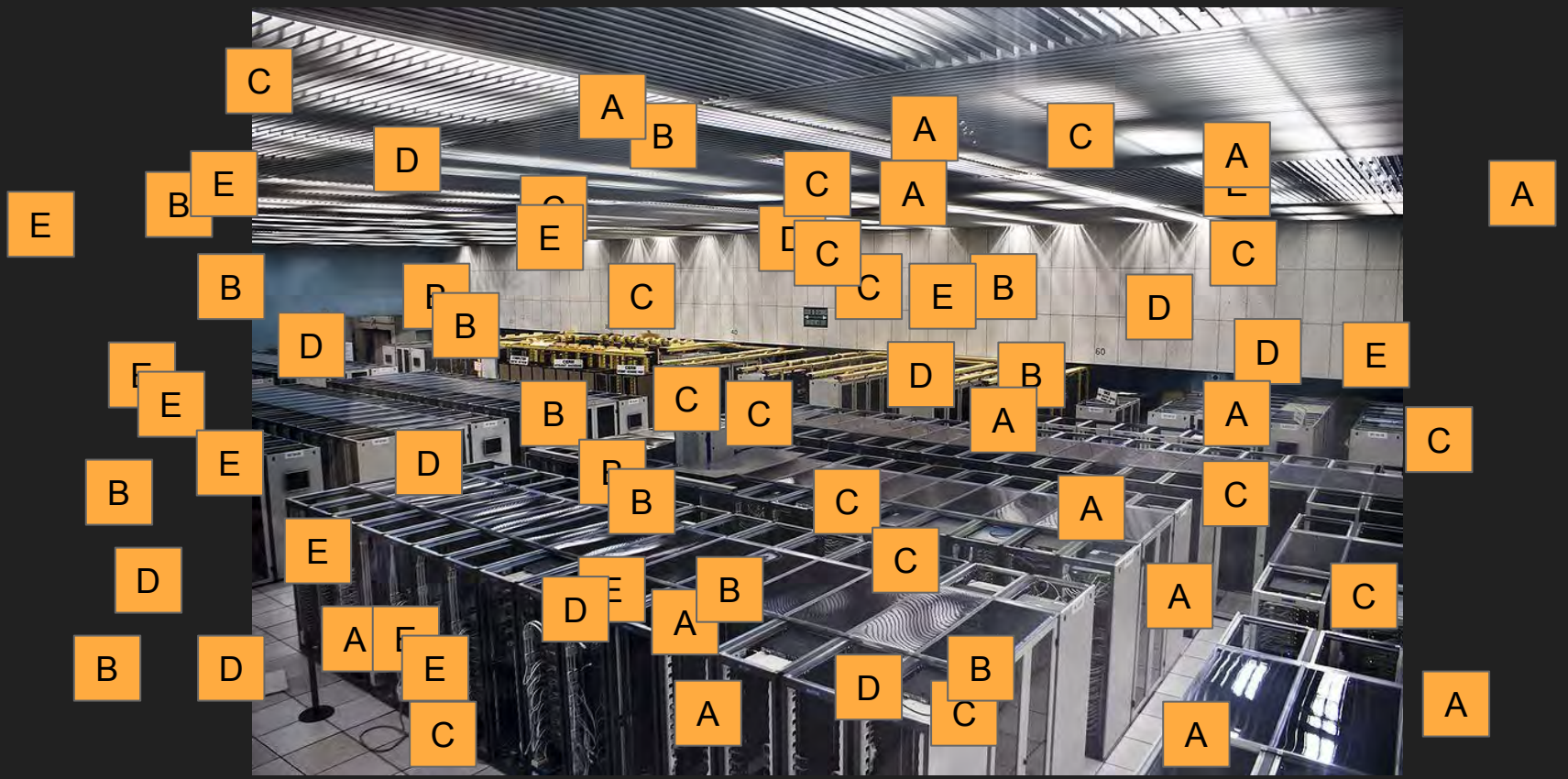
This is a Server Farm



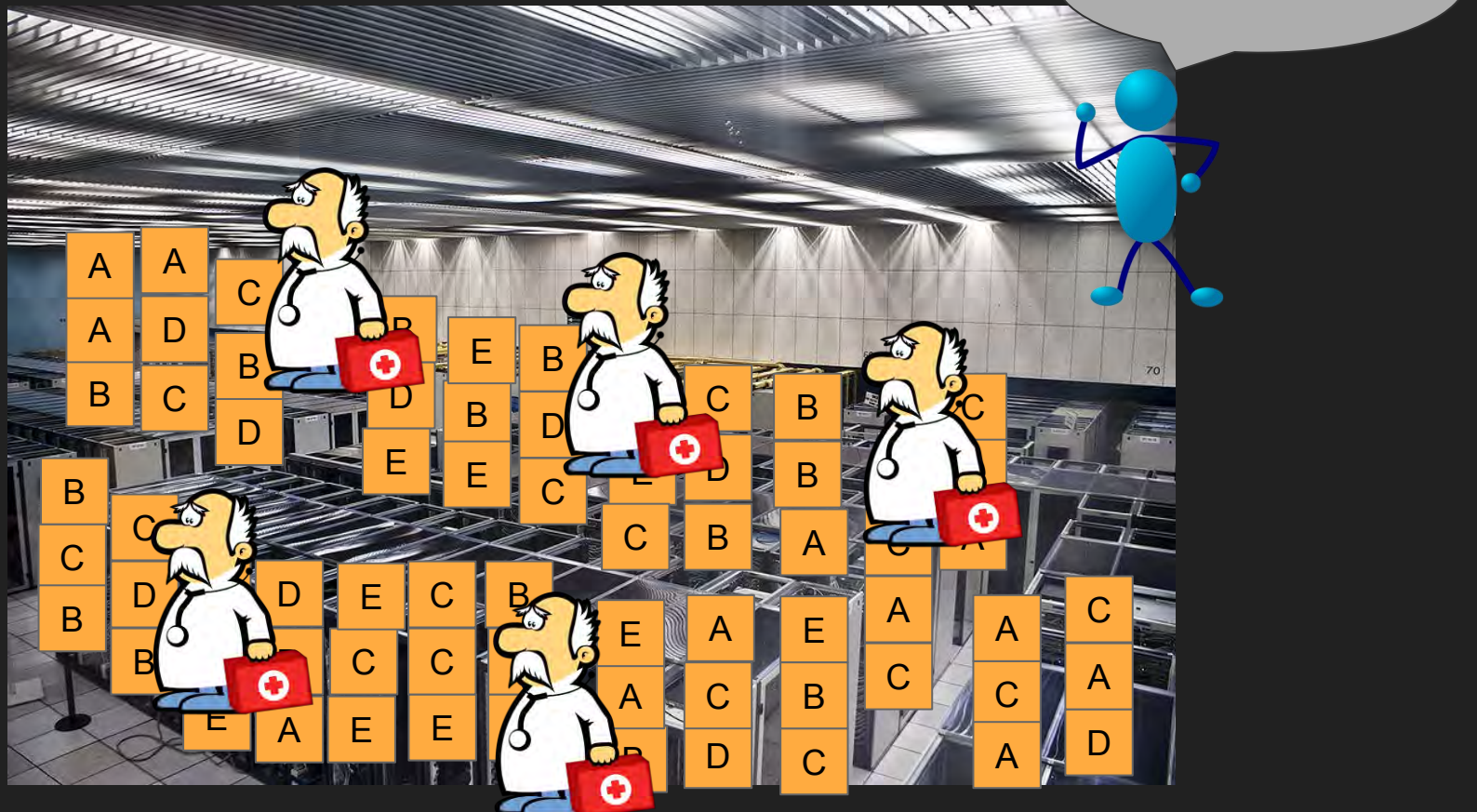
These are Micro-Service Instances



We Need To Put *These* Into *These*



And This is What a Scheduler Does



Which Scheduler?

Mesos/Marathon

Nomad

Mesos/Aurora

Kubernetes

This Scheduler!

Why?

Mesos/Marathon

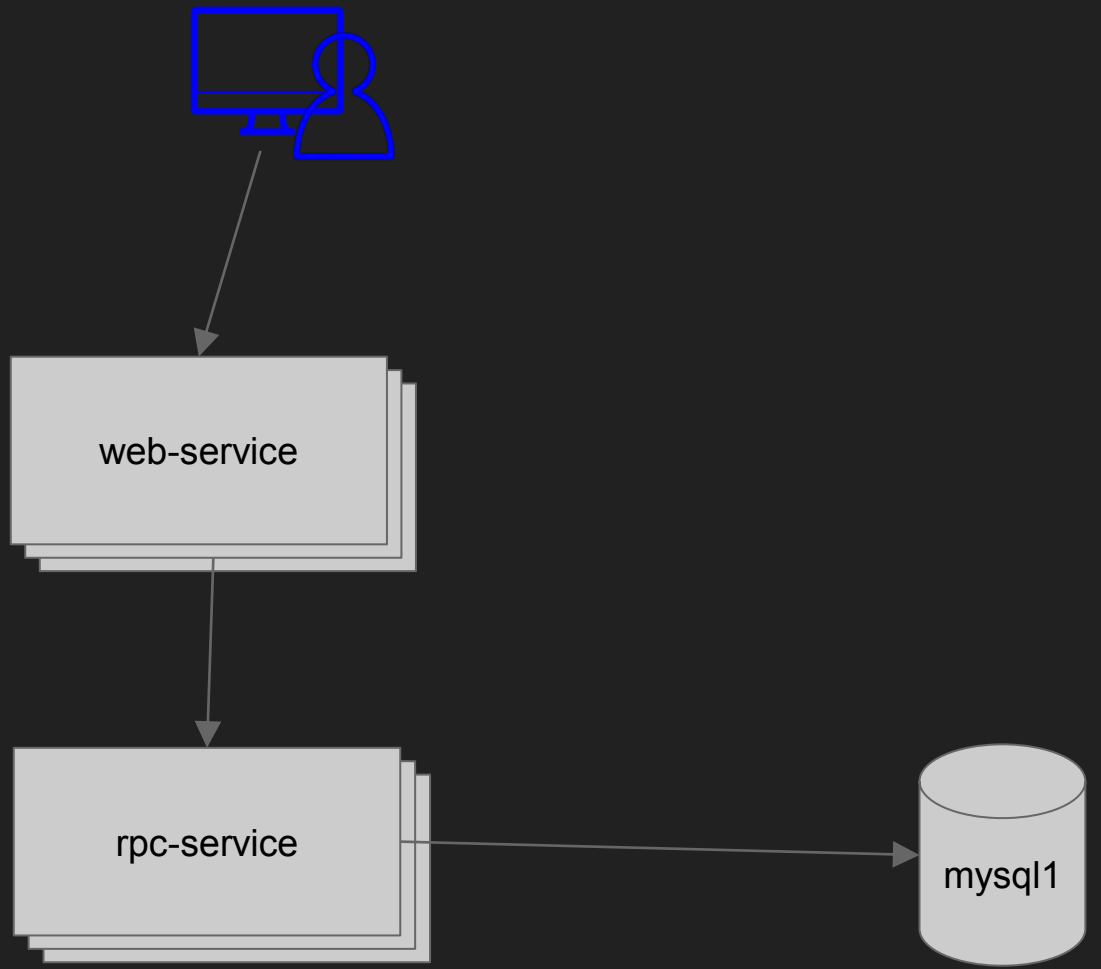
Why?

Mesos/Marathon

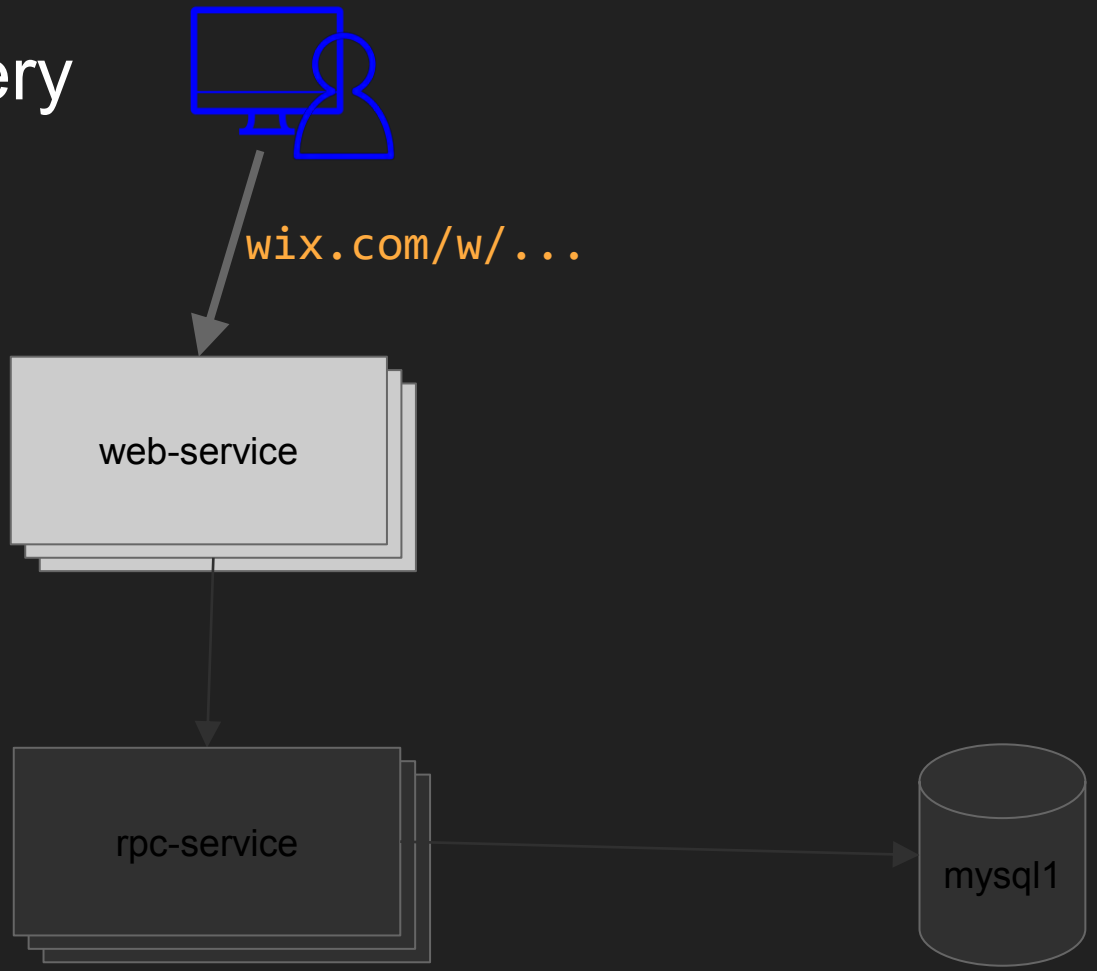
Because!


Service Discovery

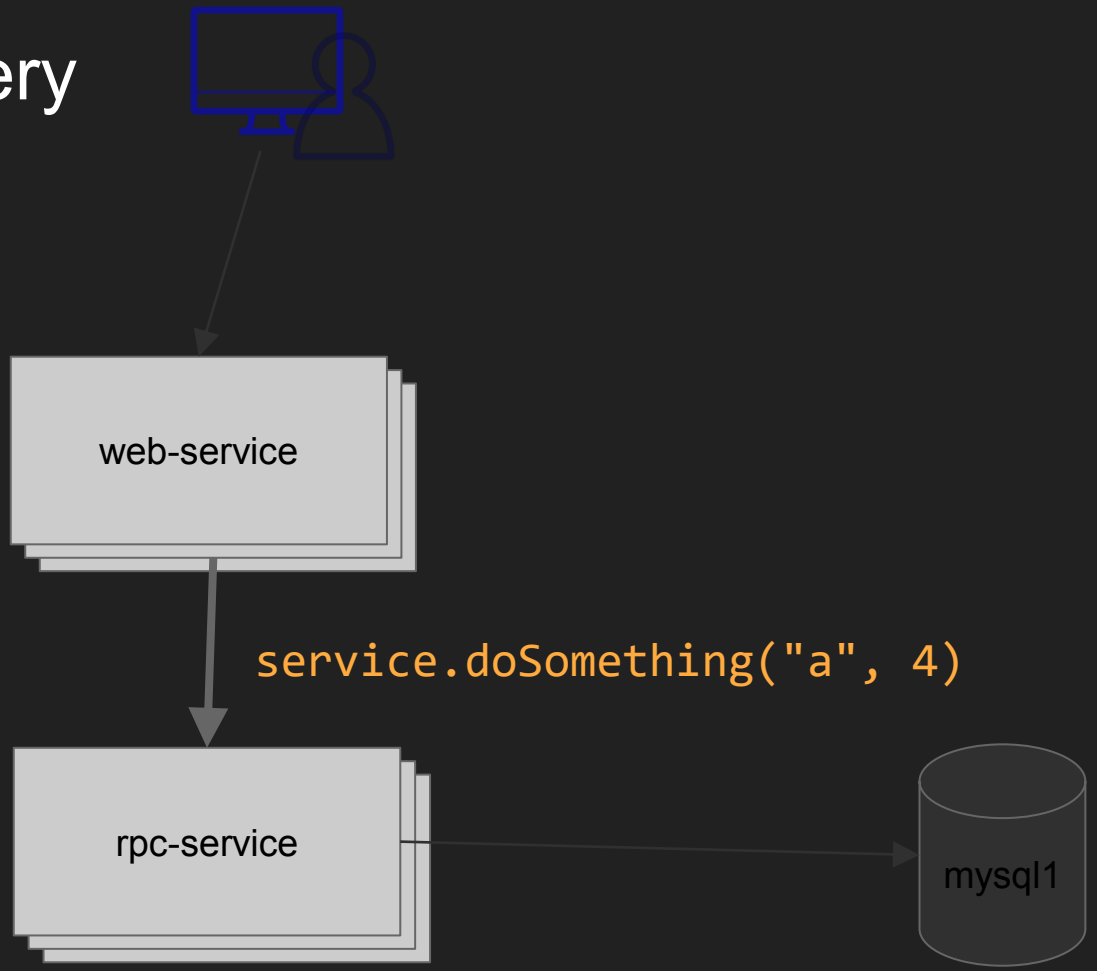
Three Kinds



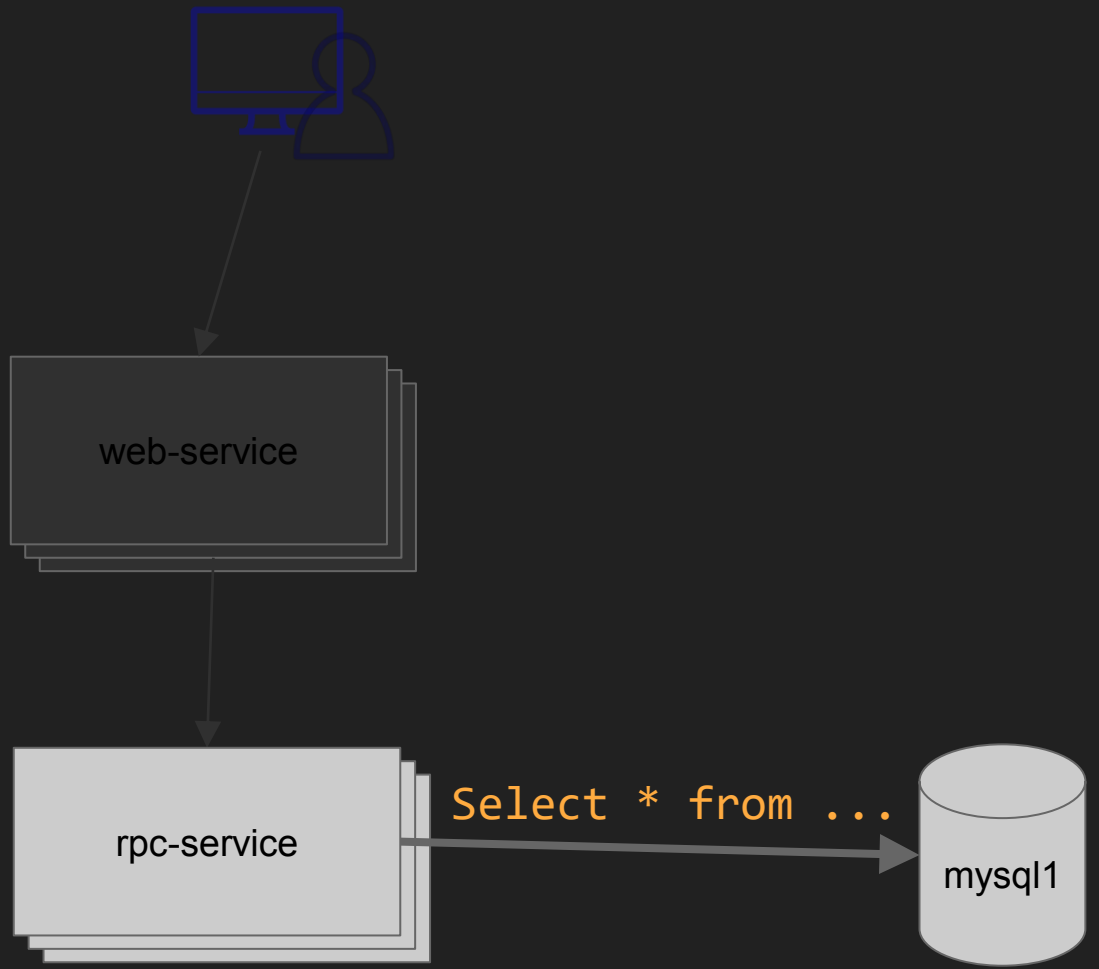
Web Service Discovery



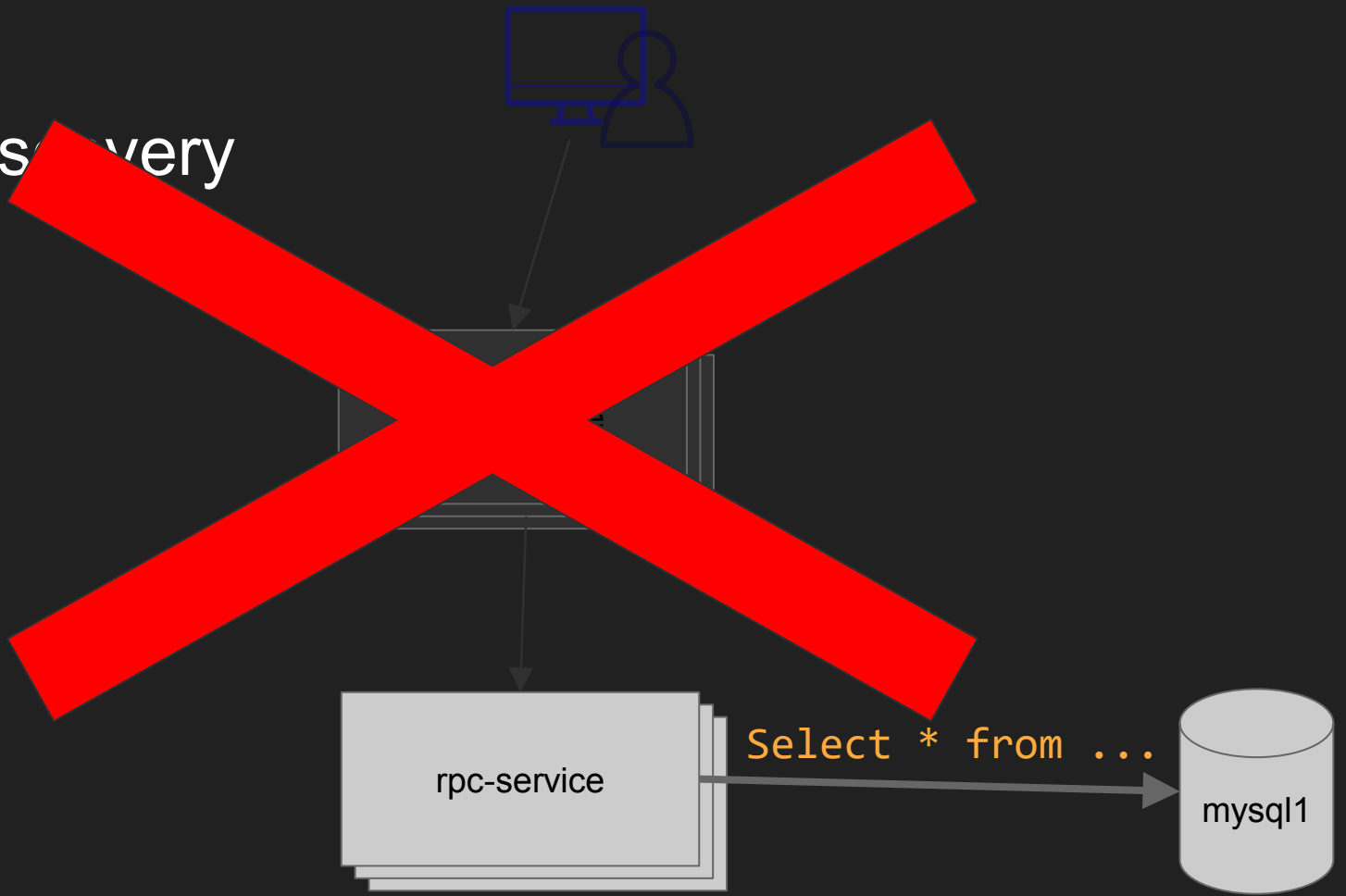
RPC Service Discovery



3rd Party
Service Discovery

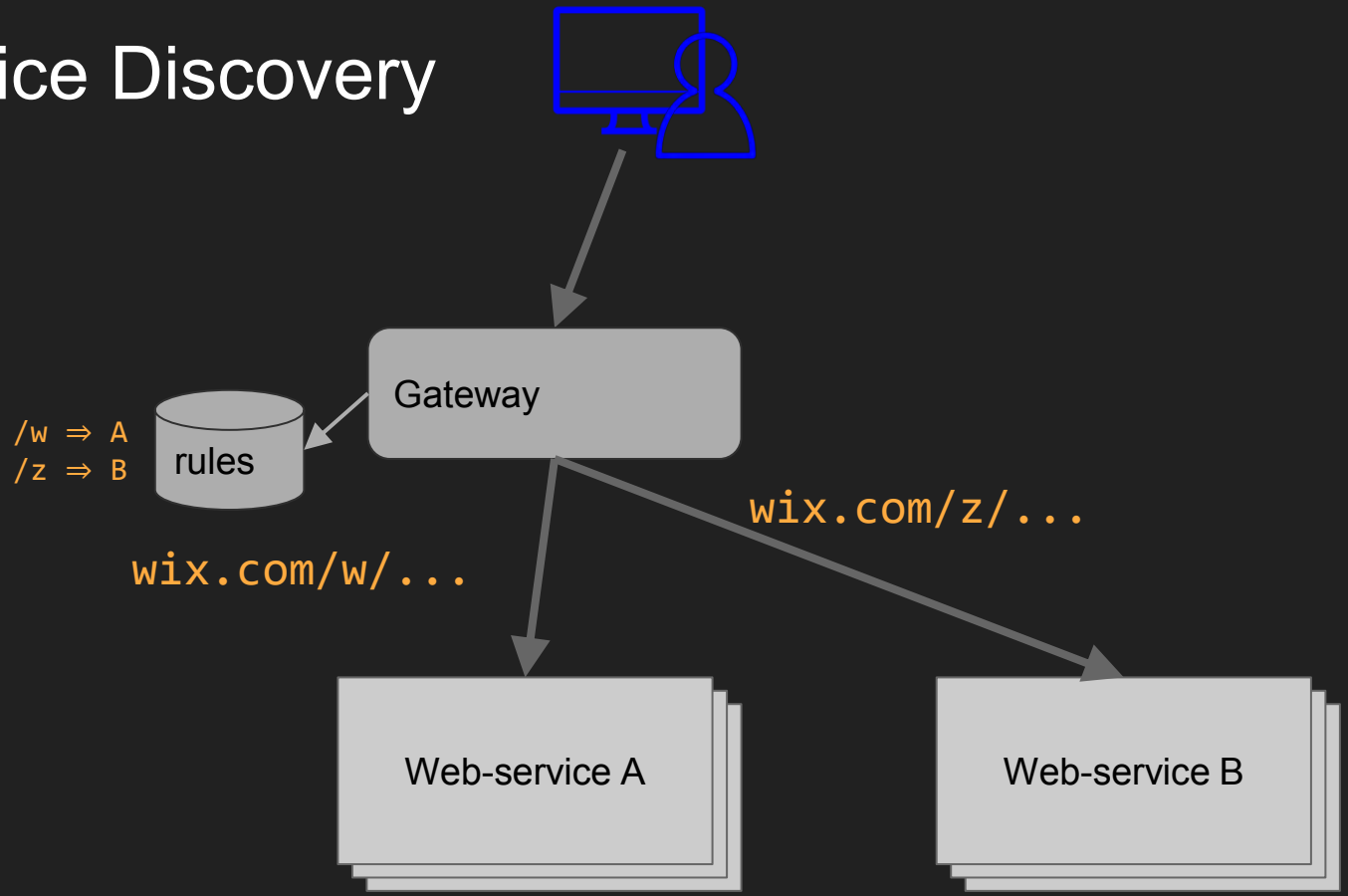


3rd Party
Service Dis very

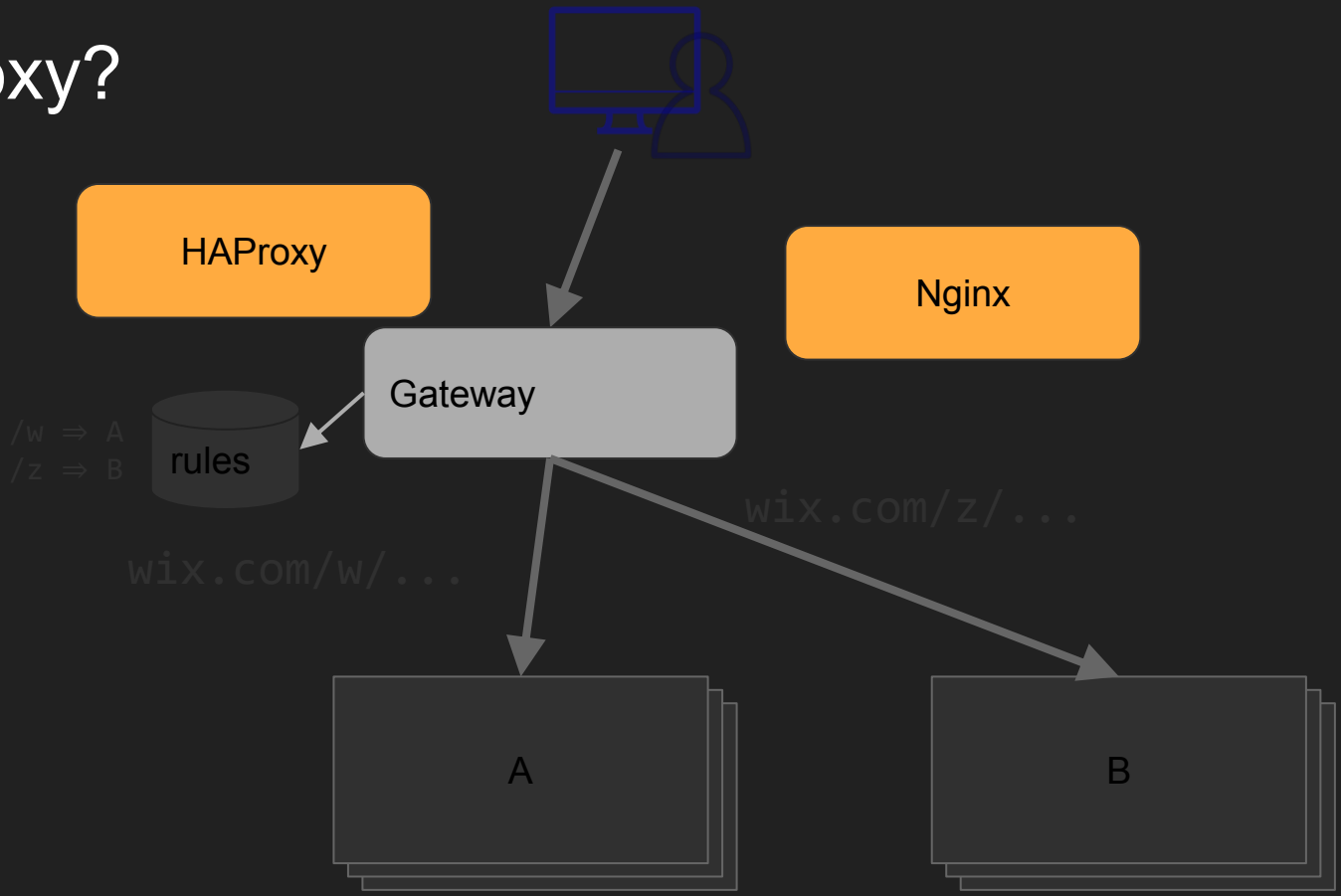


How To Implement Web Service Discovery

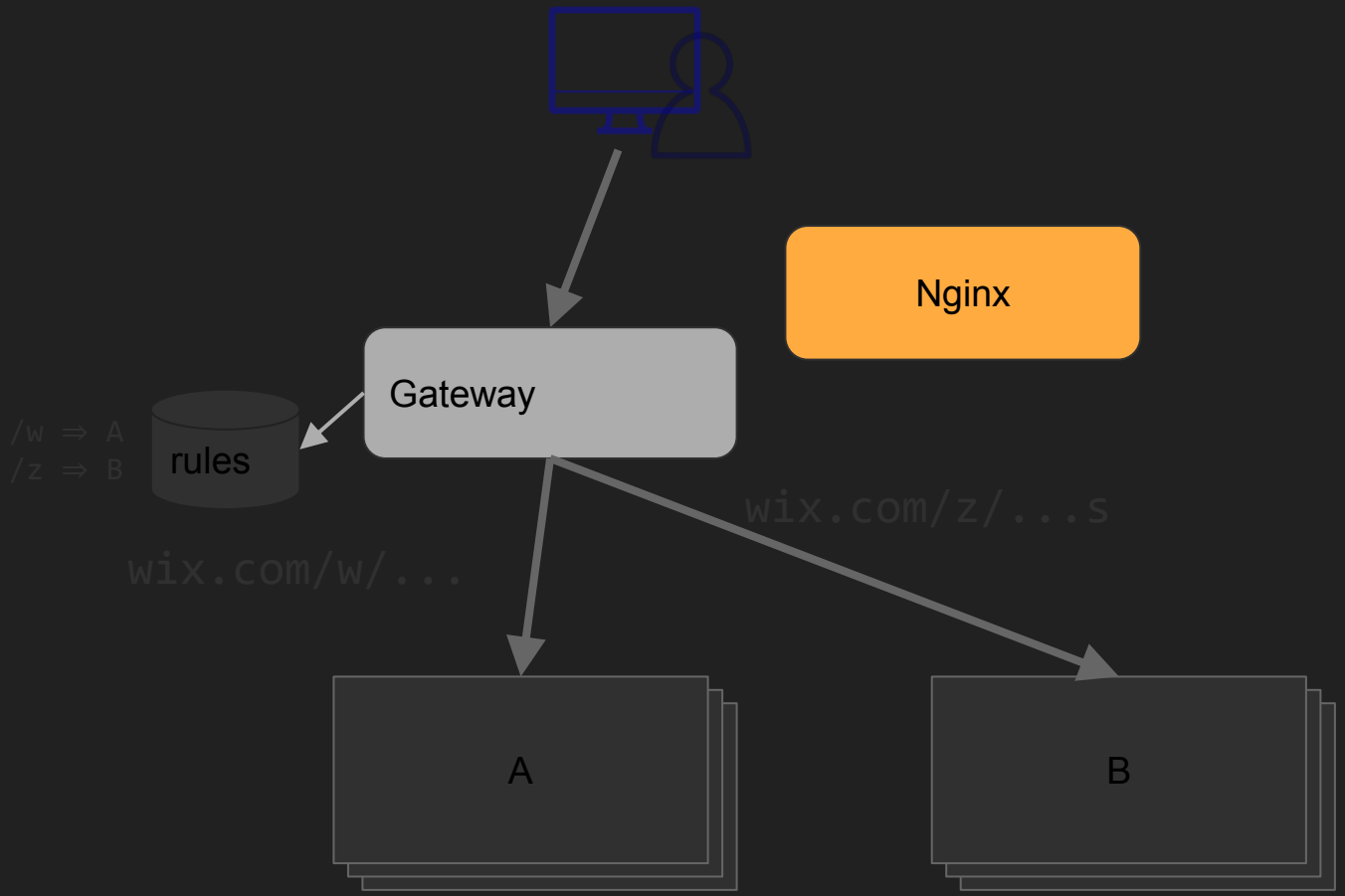
Web Service Discovery



Which Proxy?

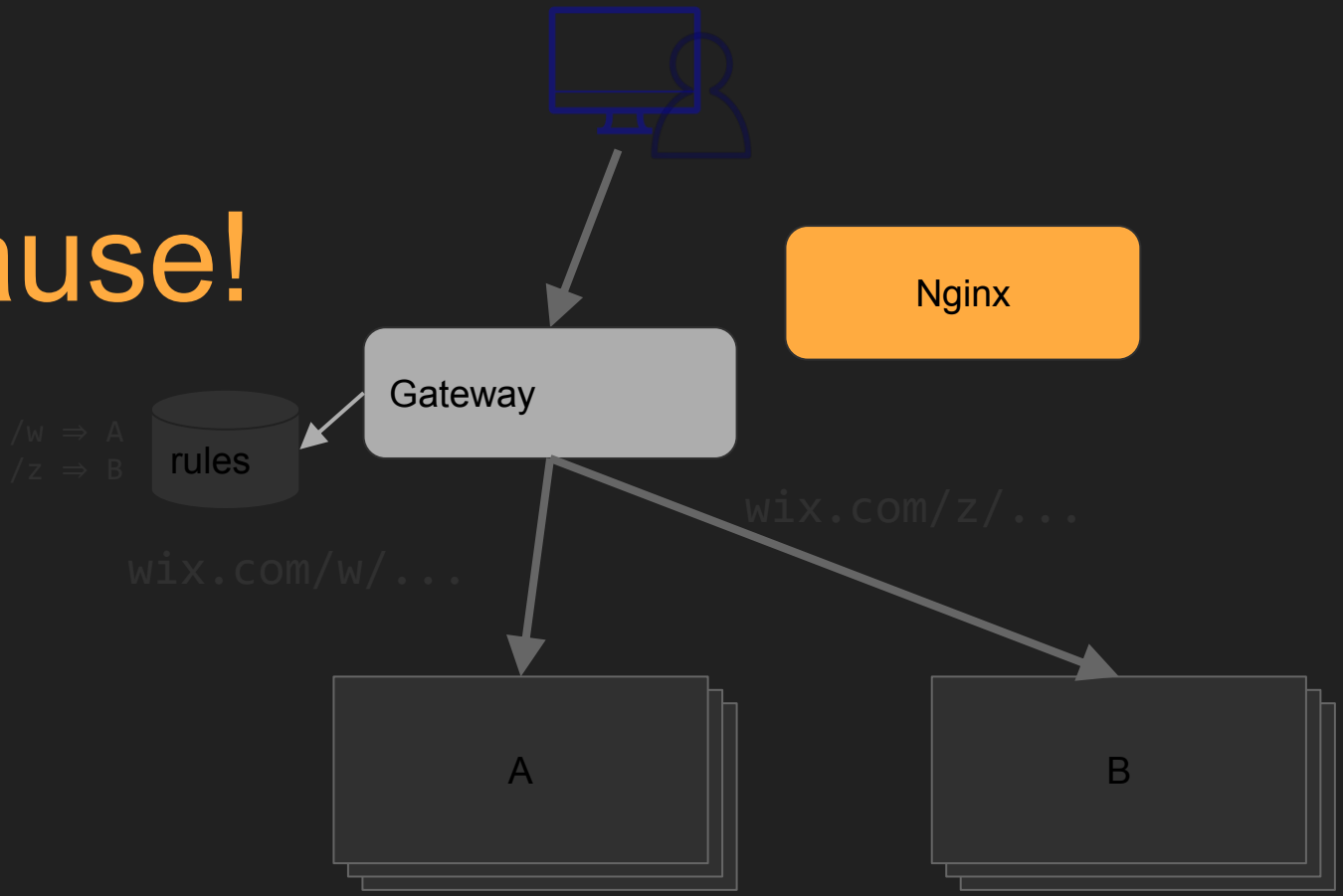


Why?

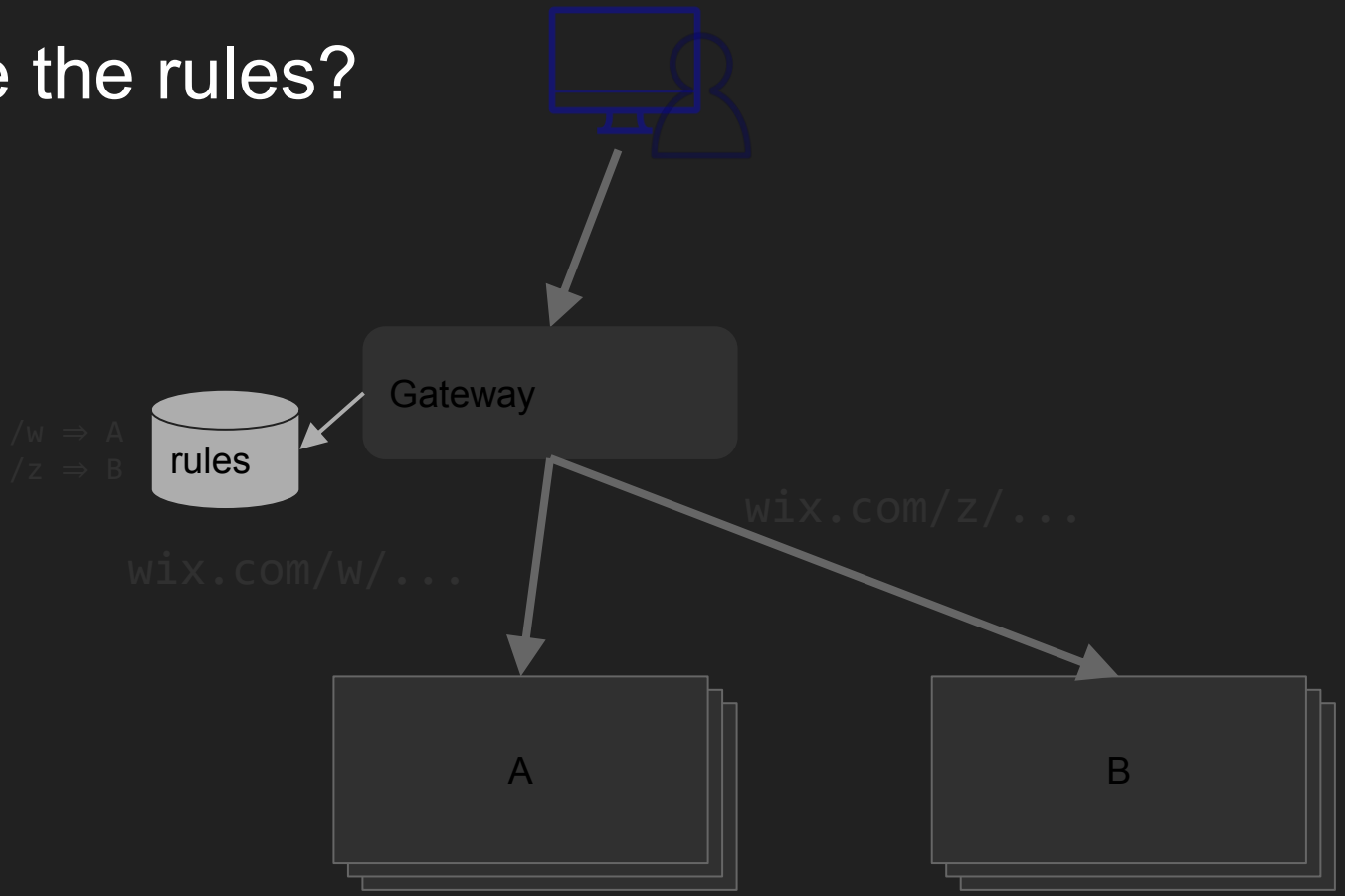


Why?

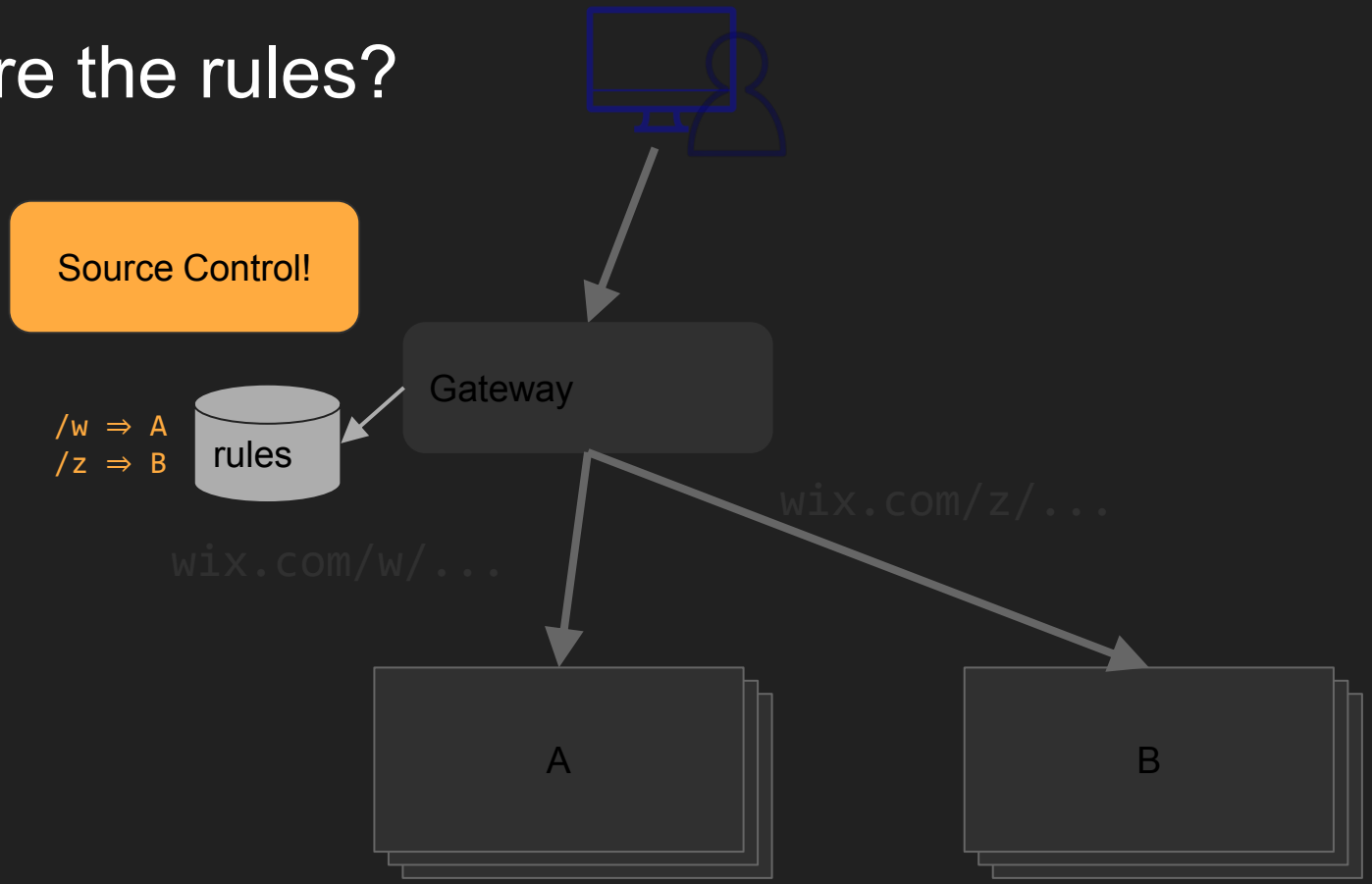
Because!



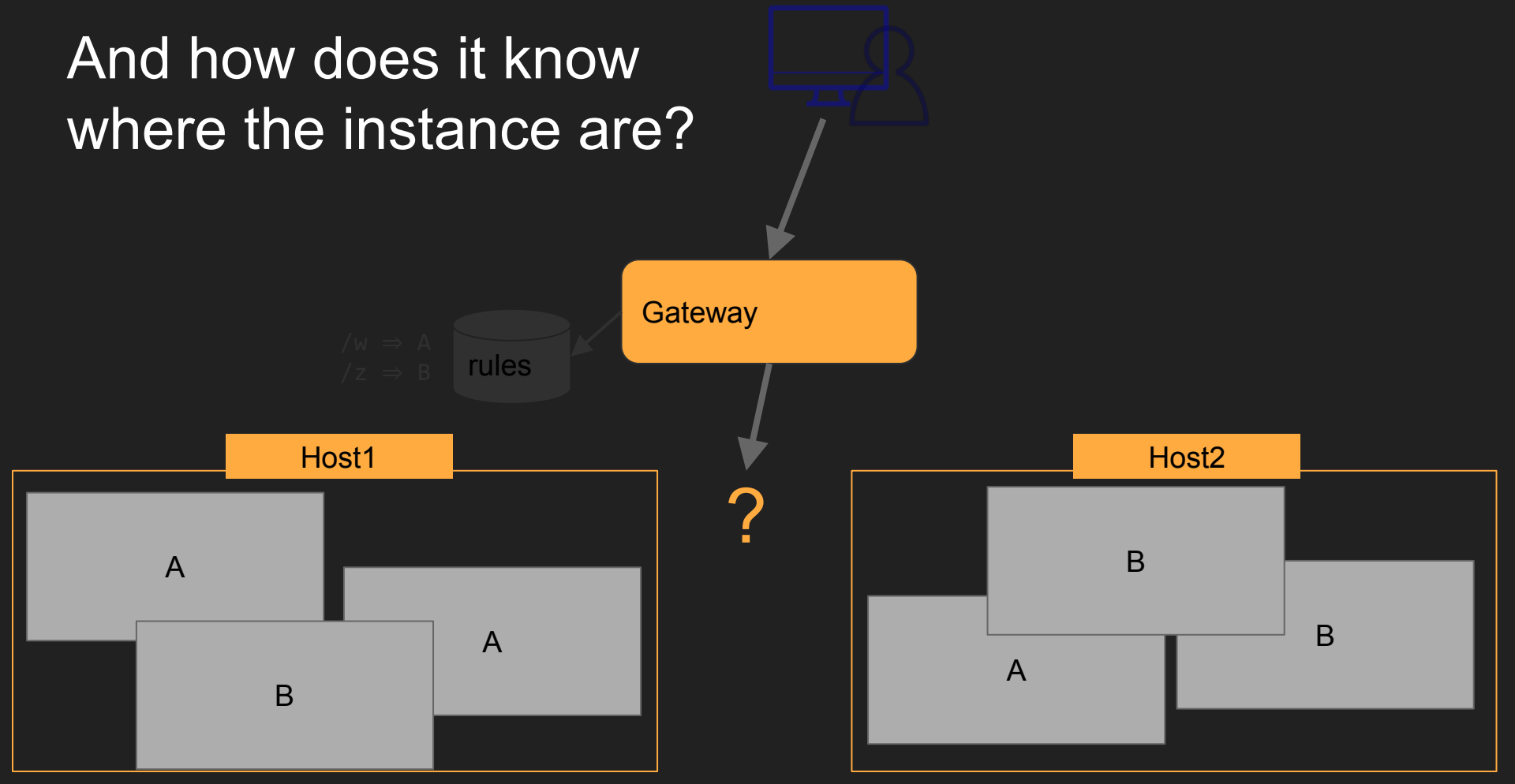
Where are the rules?



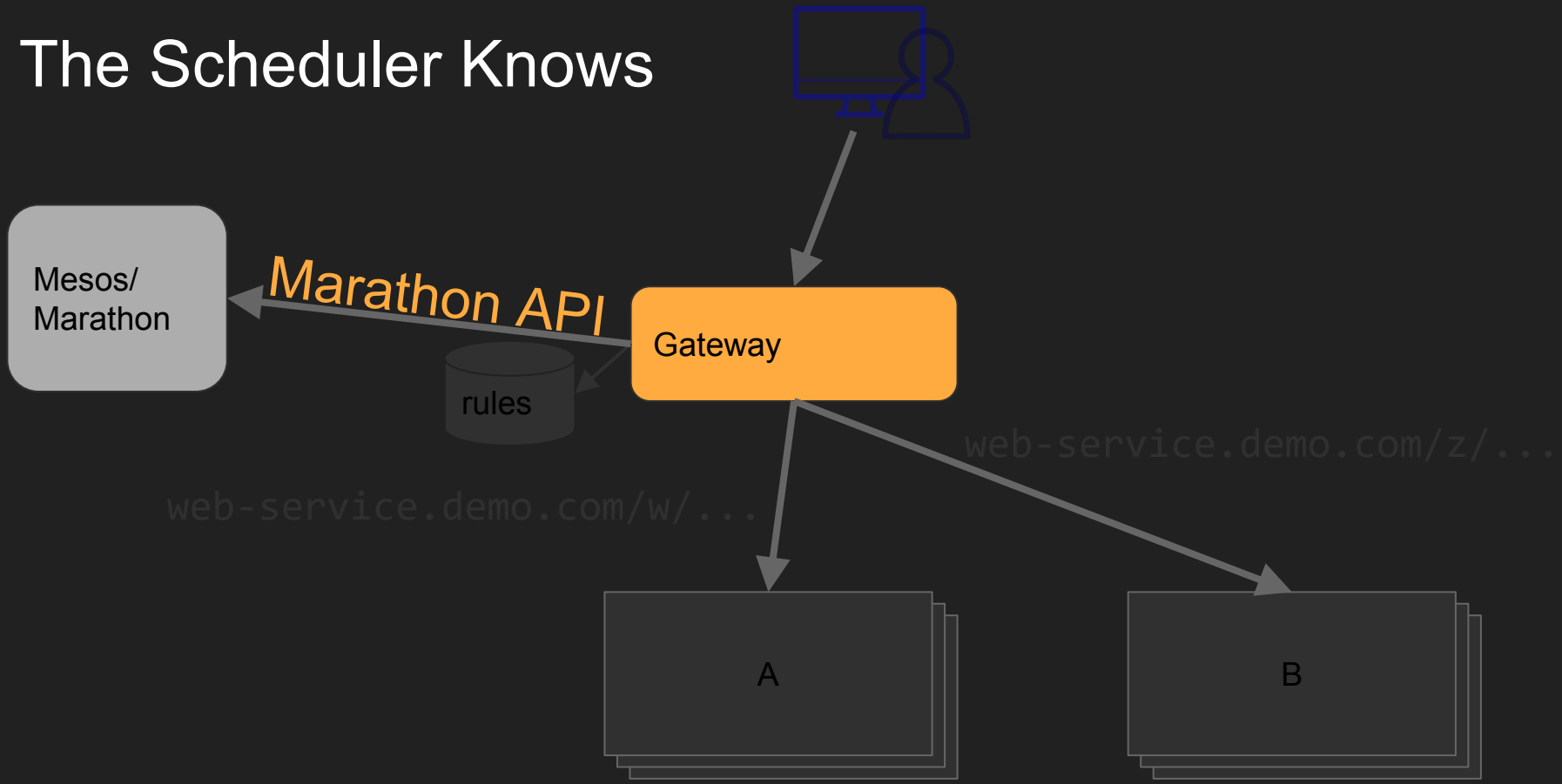
Where are the rules?



And how does it know where the instance are?

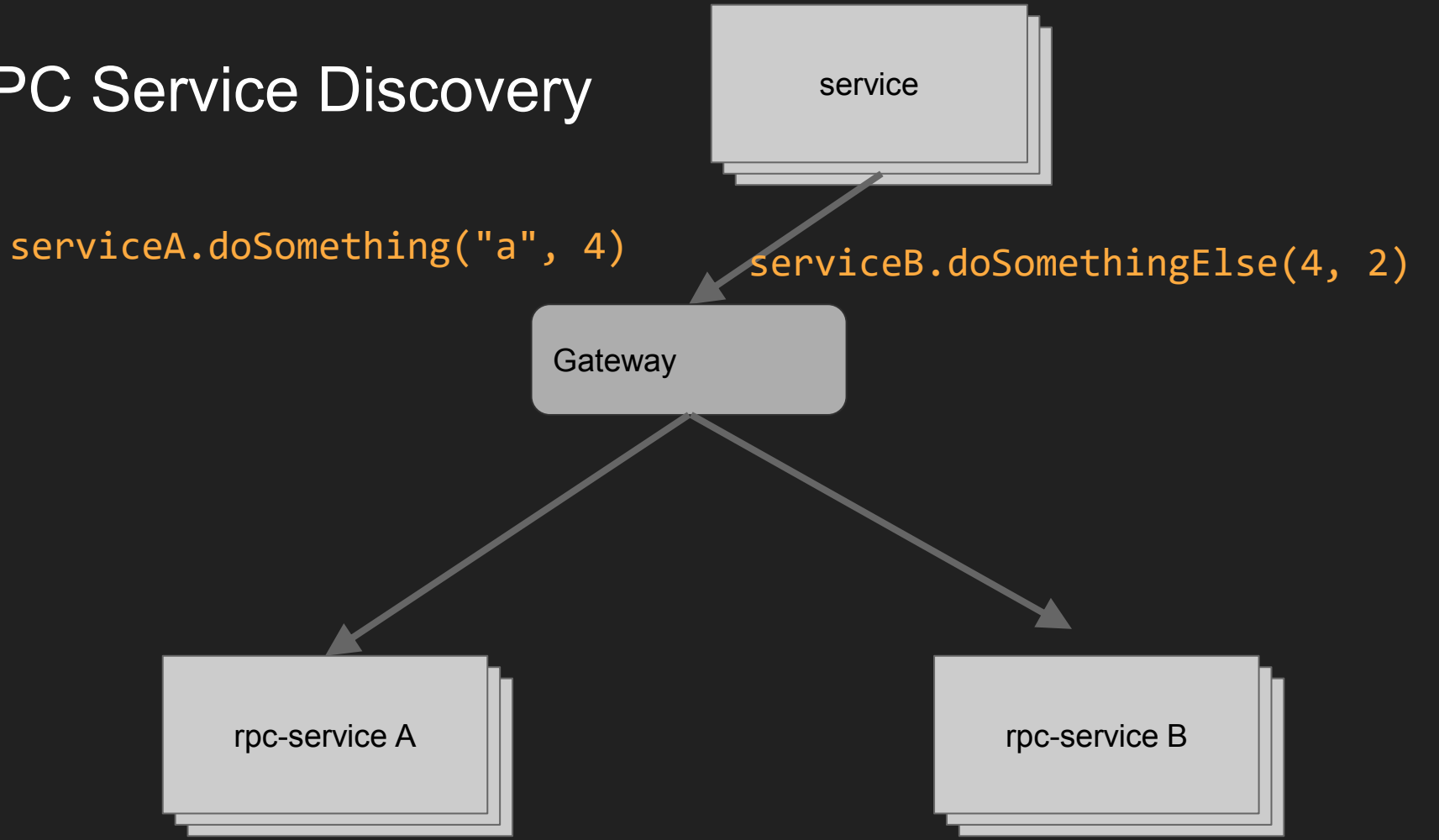


The Scheduler Knows



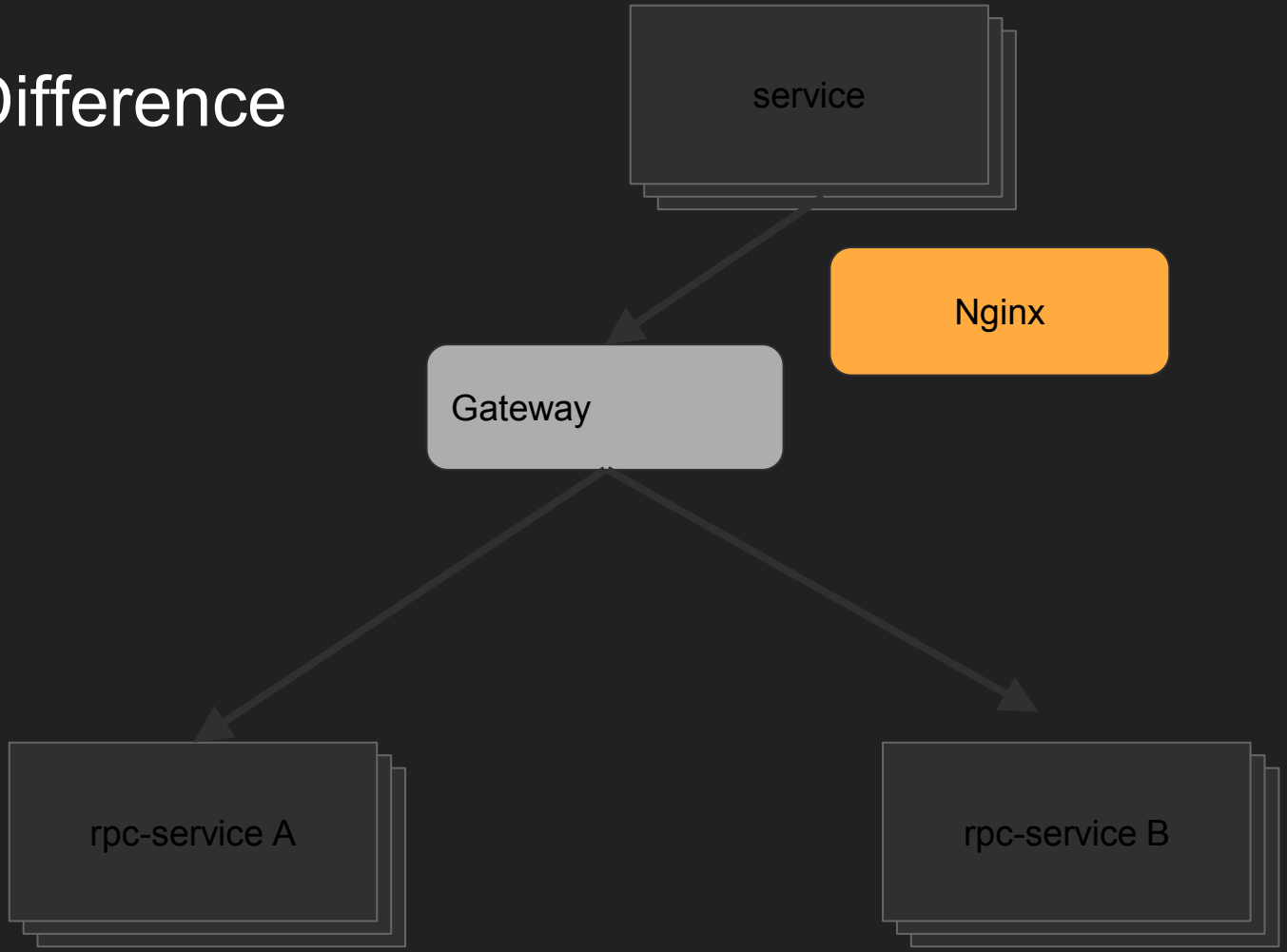
How To Implement RPC Service Discovery

RPC Service Discovery

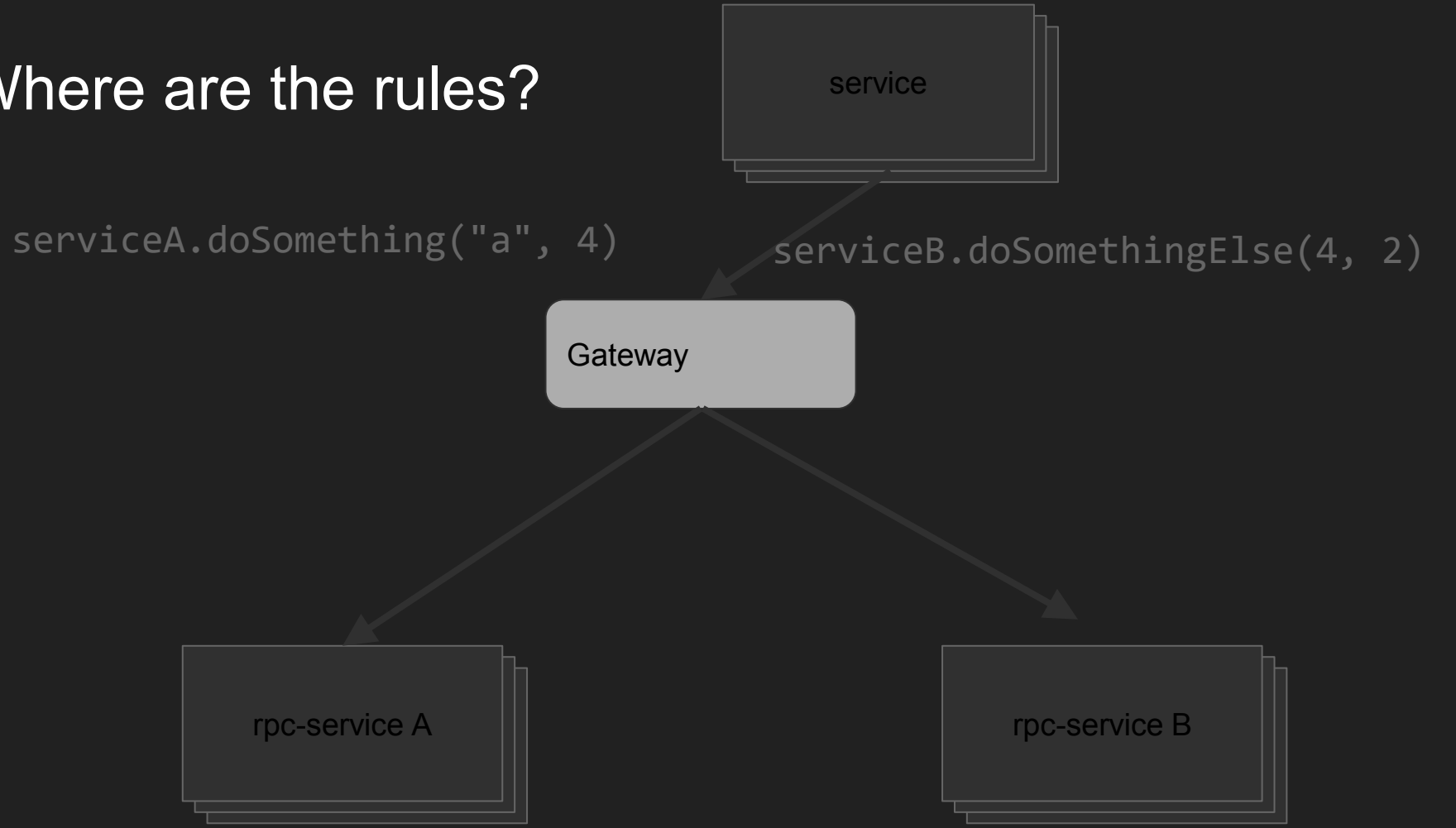



See the Resemblance?

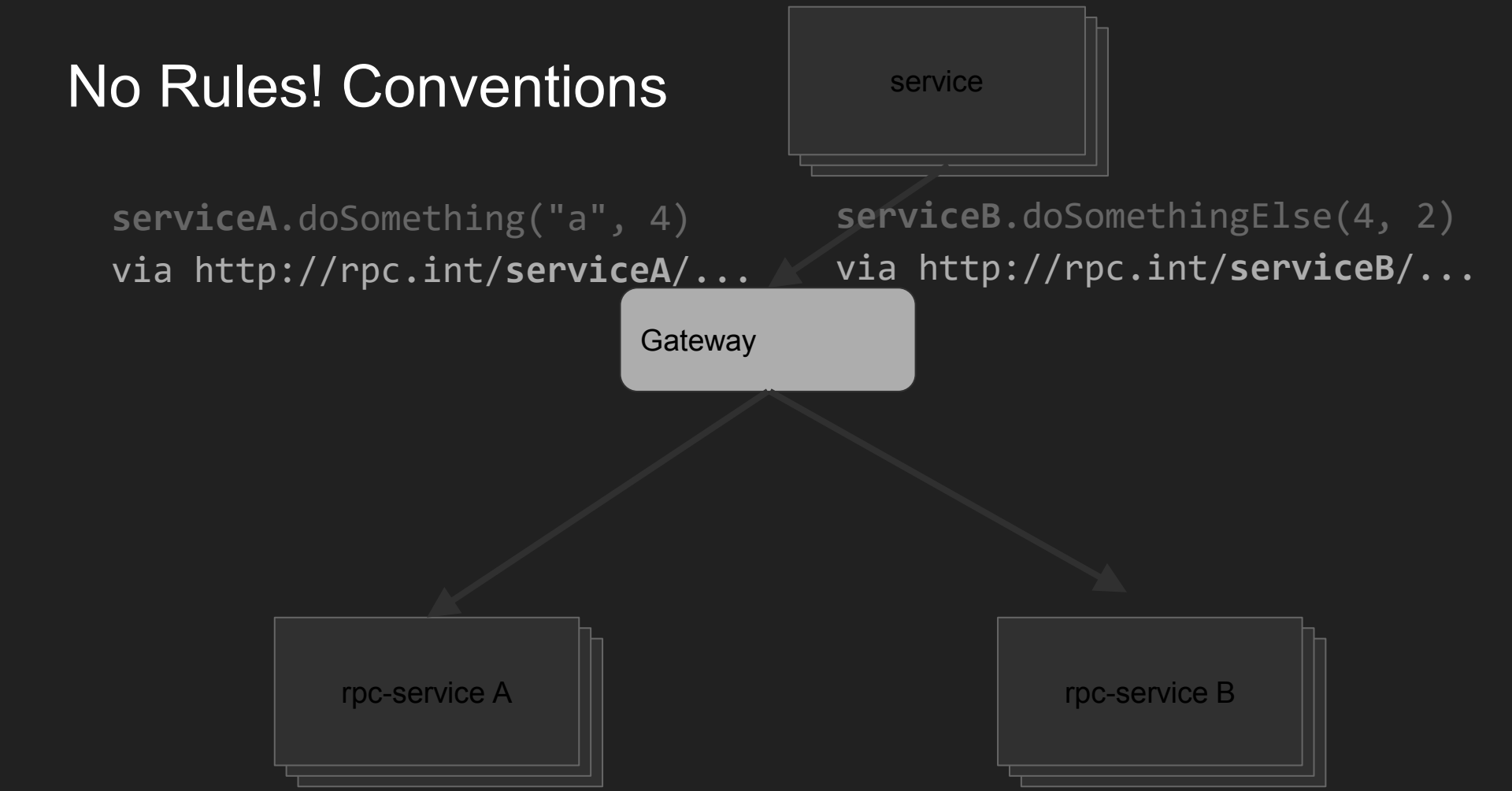
Same Difference



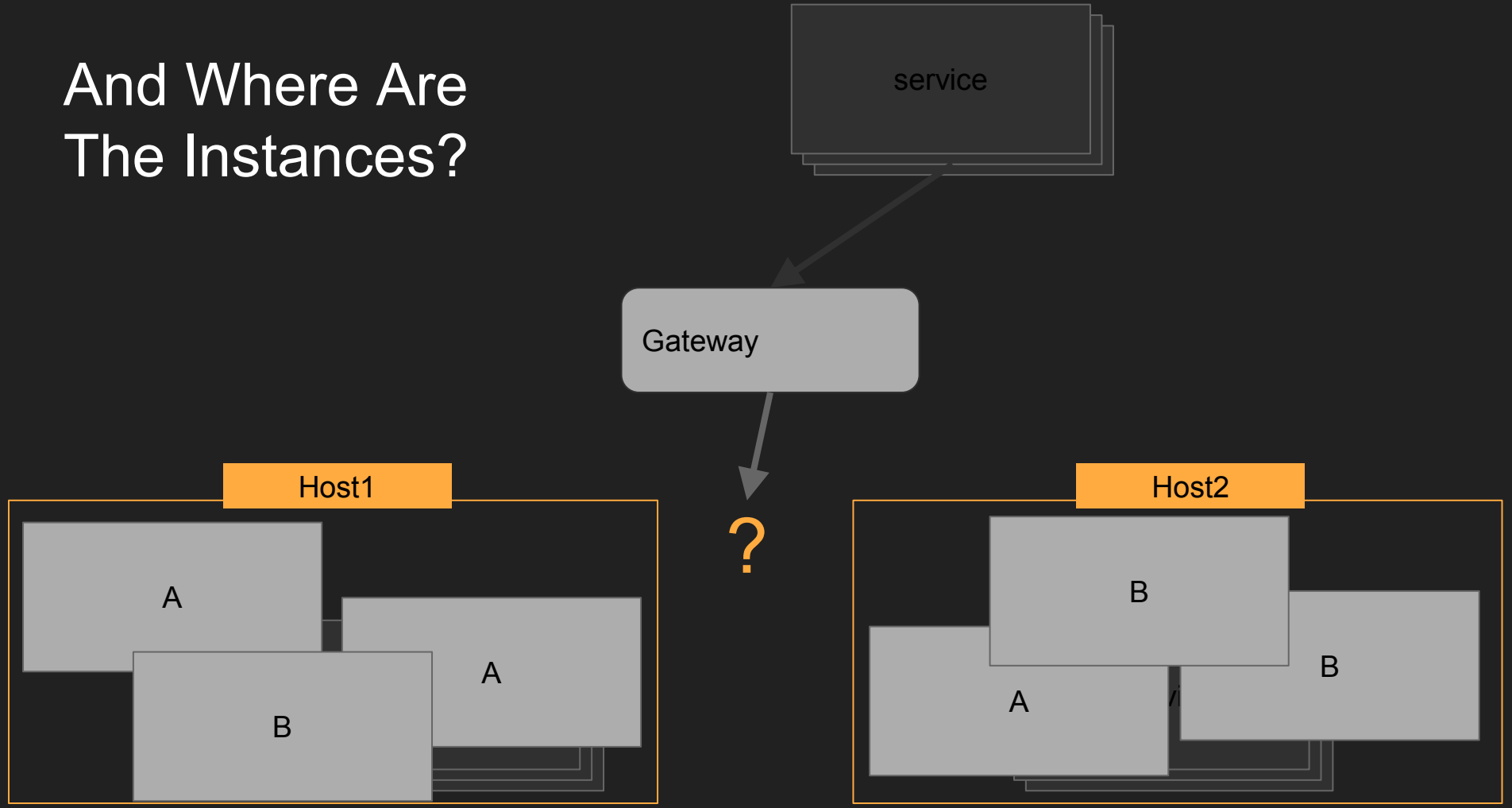
Where are the rules?



No Rules! Conventions



And Where Are The Instances?



Same Difference!

Let’s Bind This All

apollo deploy A -n 2

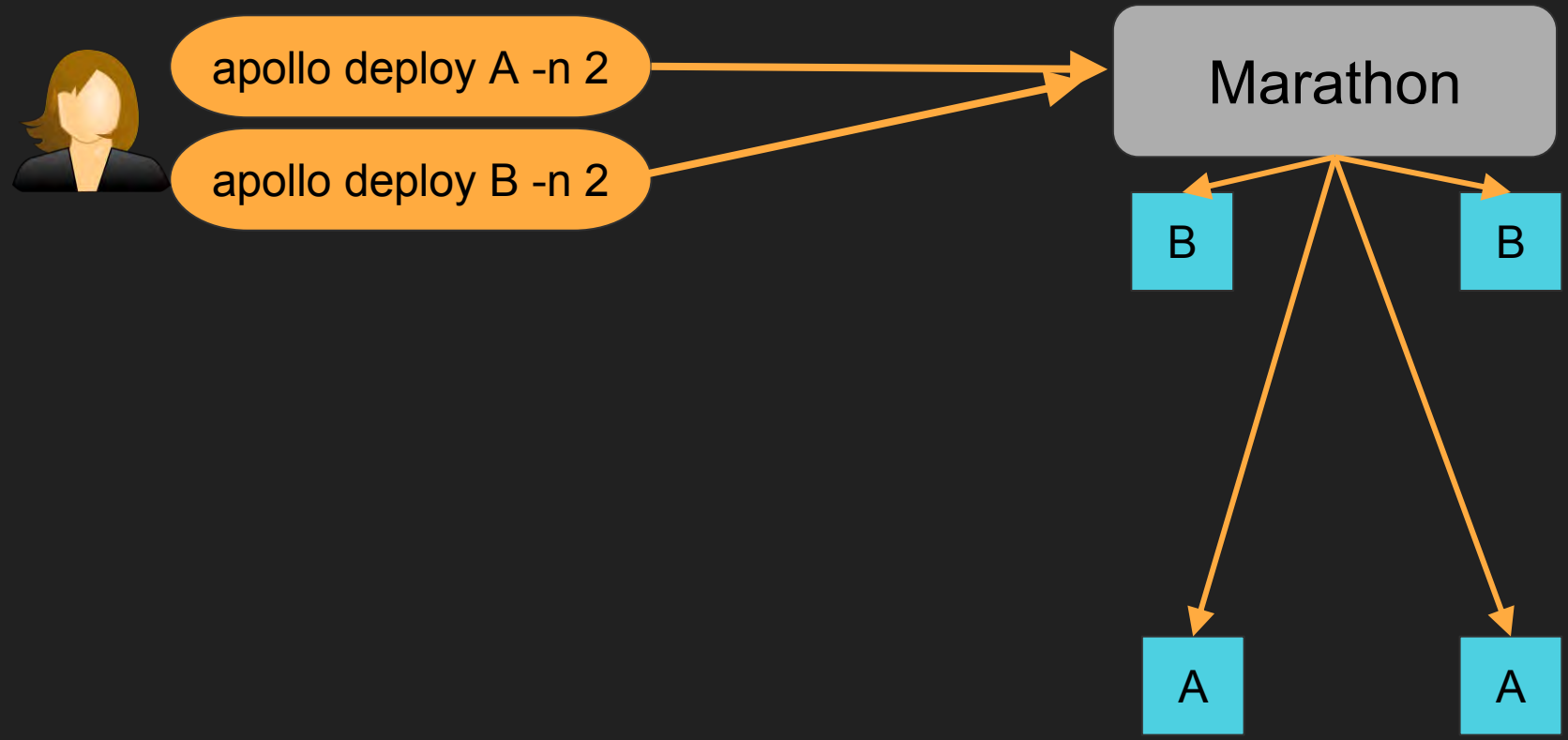
apollo deploy B -n 2

apollo deploy A -n 2

apollo deploy B -n 2

Marathon

- Please create service A
- I need two instances of this service available at all times
- This is the docker image you need to deploy


apollo deploy A -n 2
apollo deploy B -n 2
Marathon
B
B
A
A

apollo deploy A -n 2

apollo deploy B -n 2

Marathon

B

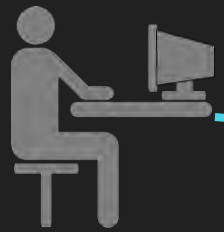
B

A

C

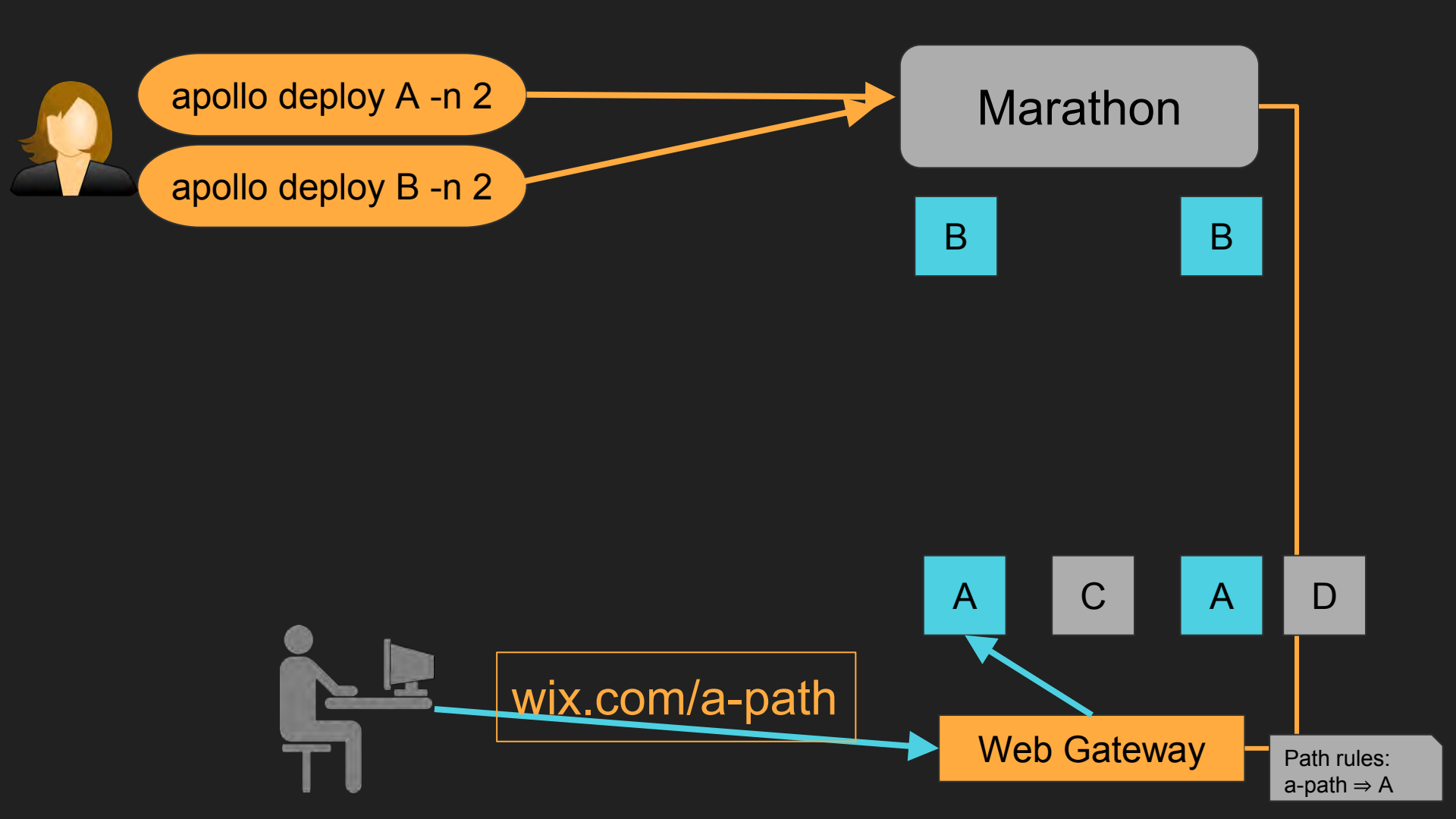
A

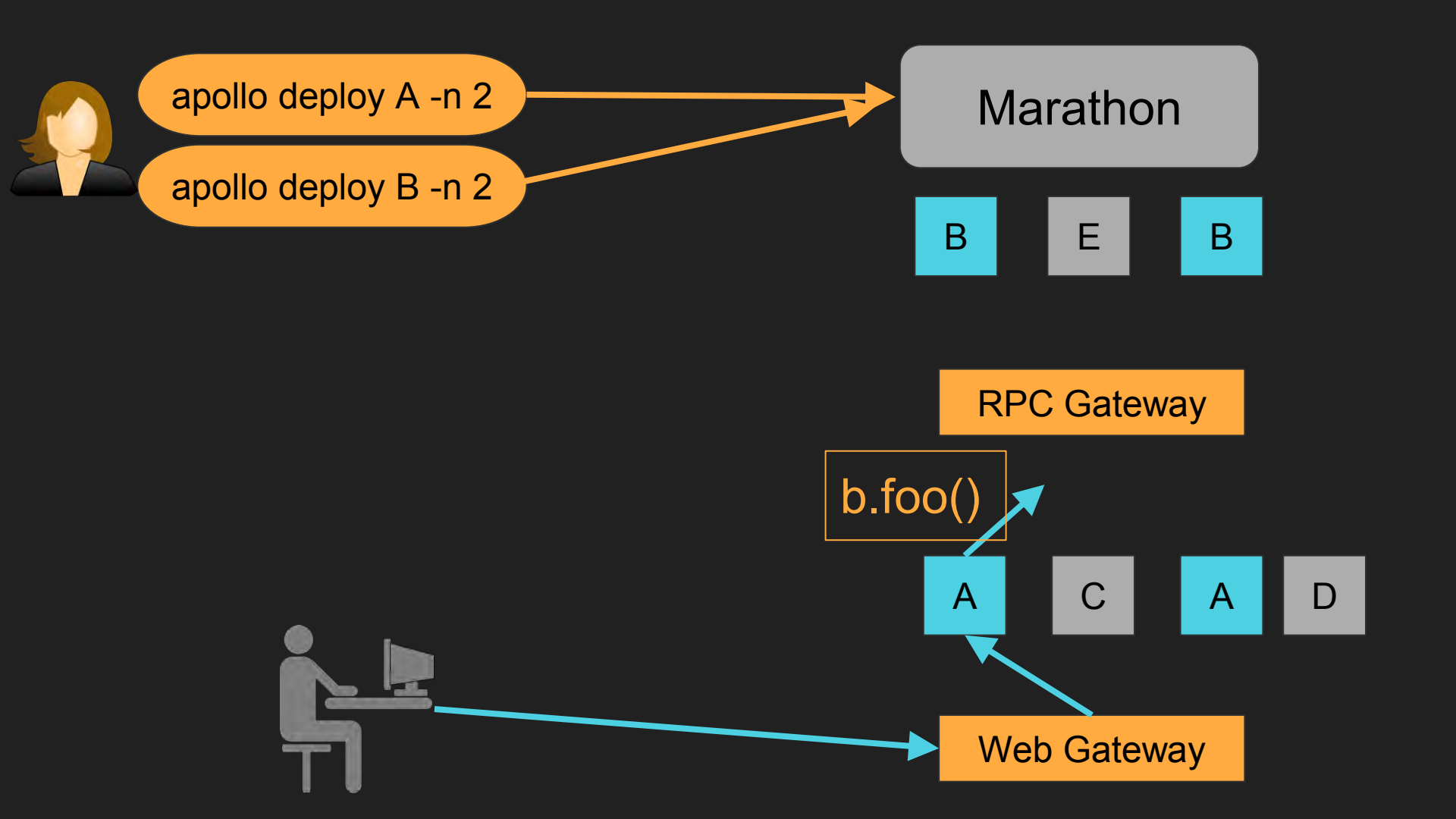
D

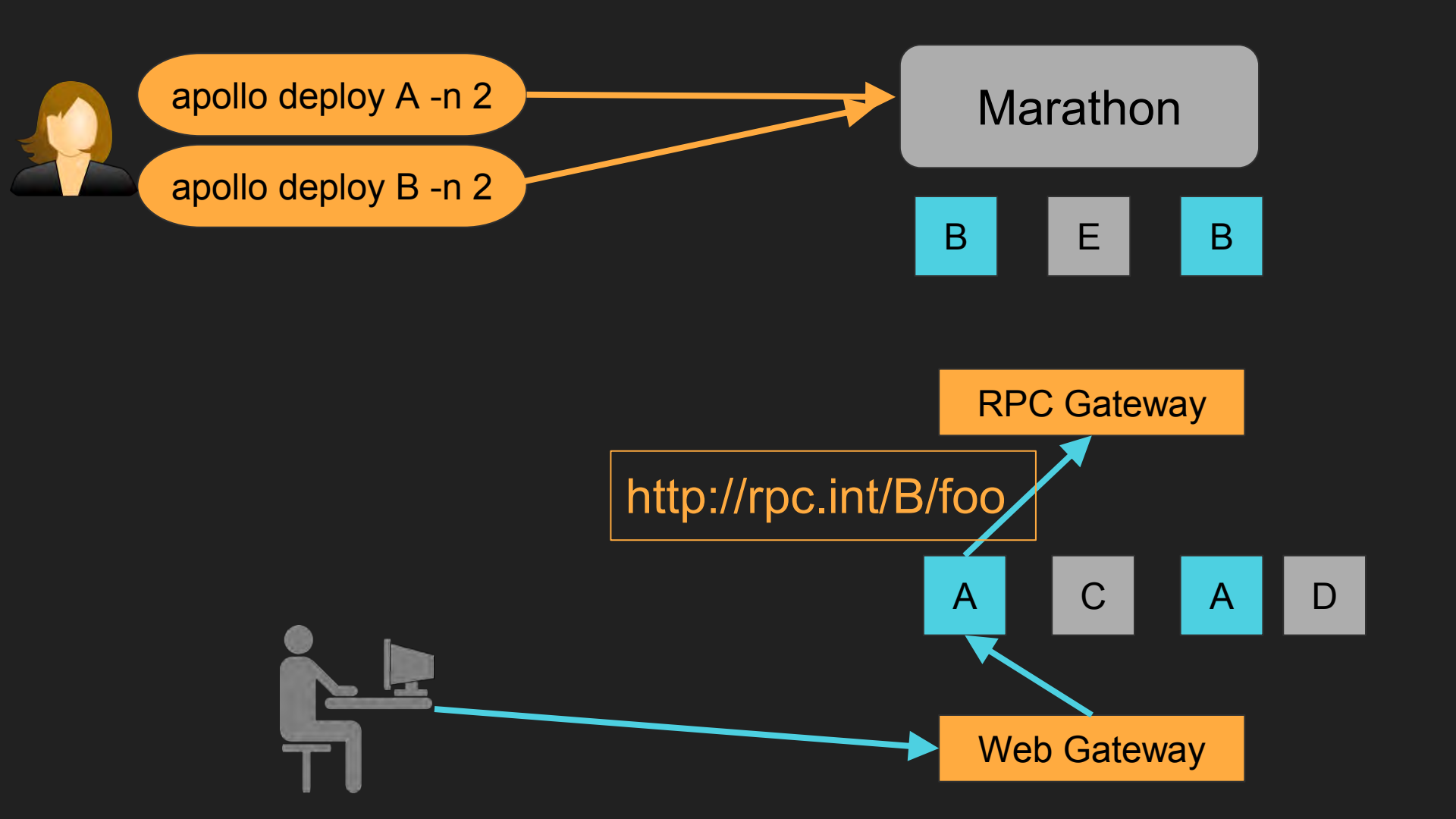


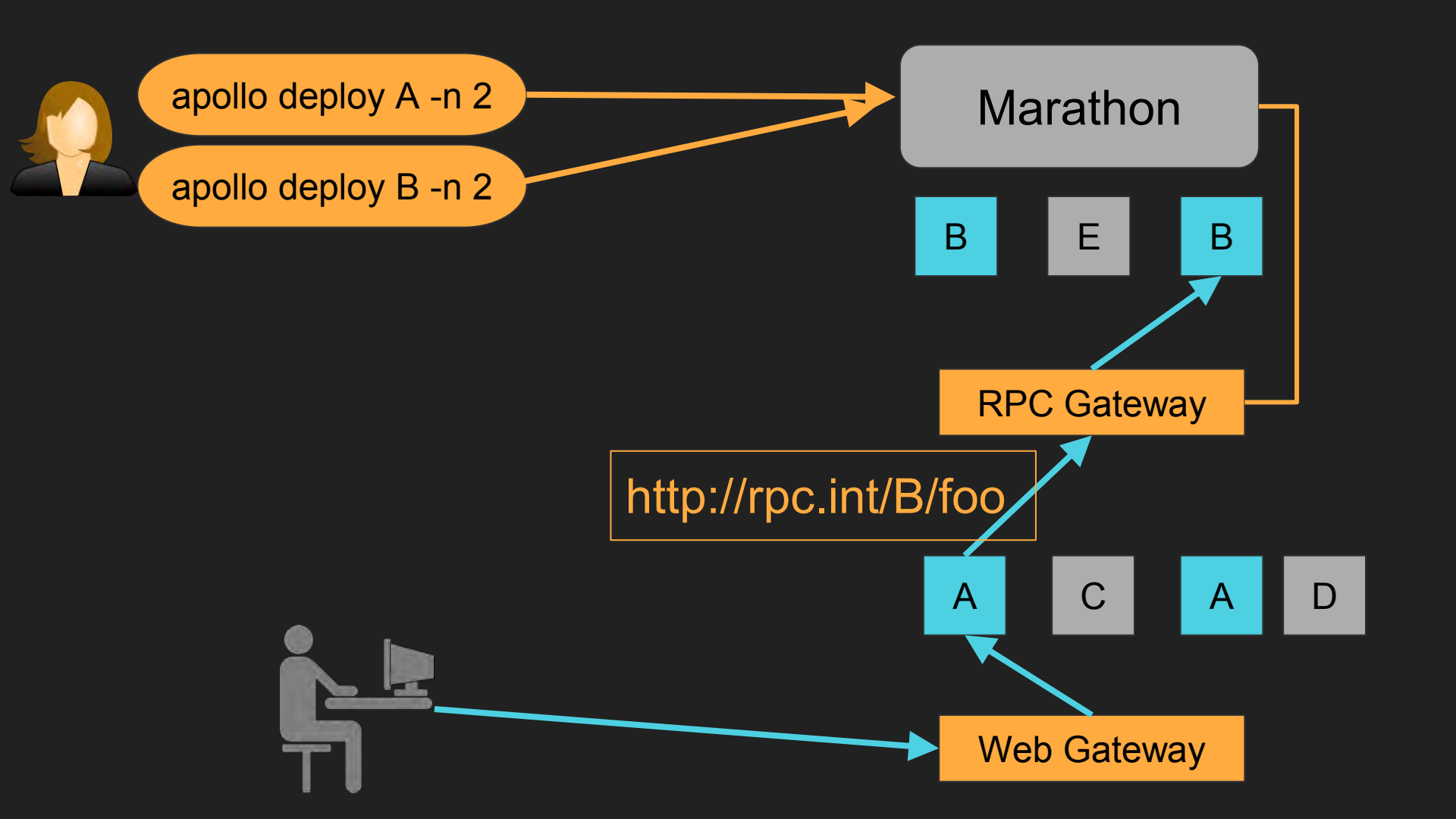
wix.com/a-path

Web Gateway


apollo deploy A -n 2
apollo deploy B -n 2
Marathon
B
B
A
C
A
D
wix.com/a-path
Web Gateway
Path rules:
a-path ⇒ A


apollo deploy A -n 2
apollo deploy B -n 2
Marathon
B
E
B
RPC Gateway
b.foo()
A
C
A
D
Web Gateway


apollo deploy A -n 2
apollo deploy B -n 2
Marathon
B
E
B
RPC Gateway
http://rpc.int/B/foo
A
C
A
D
Web Gateway


apollo deploy A -n 2
apollo deploy B -n 2
Marathon
B
E
B
RPC Gateway
http://rpc.int/B/foo
A
C
A
D
Web Gateway

apollo deploy A -n 2

apollo deploy B -n 2

Marathon

B

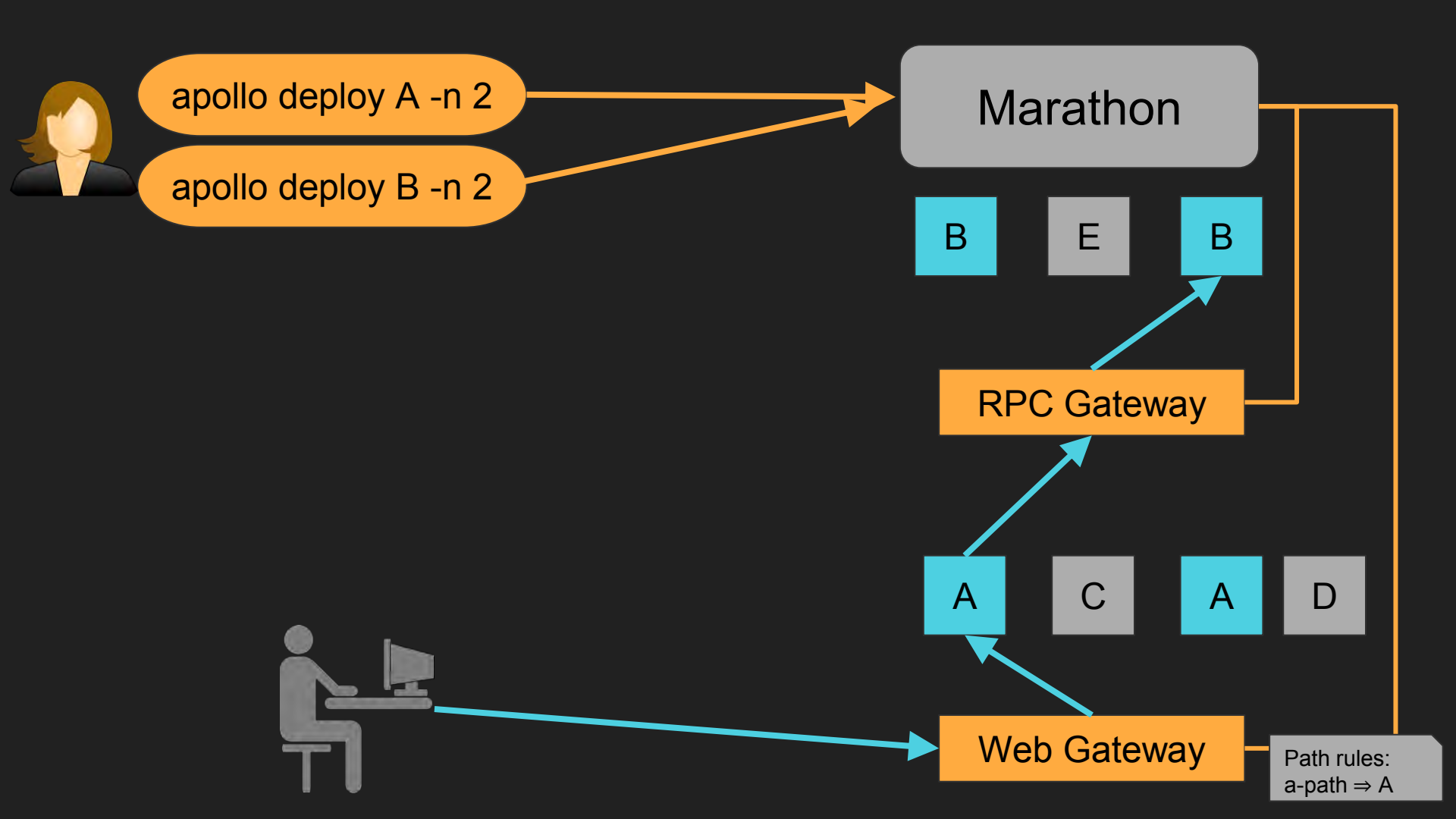
RPC Gateway

A C A D

Web Gateway

Path rules:
a-path ⇒ A

And There We Have It!


apollo deploy A -n 2
apollo deploy B -n 2
Marathon
B
E
B
RPC Gateway
A
C
A
D
Web Gateway
Path rules:
a-path ⇒ A

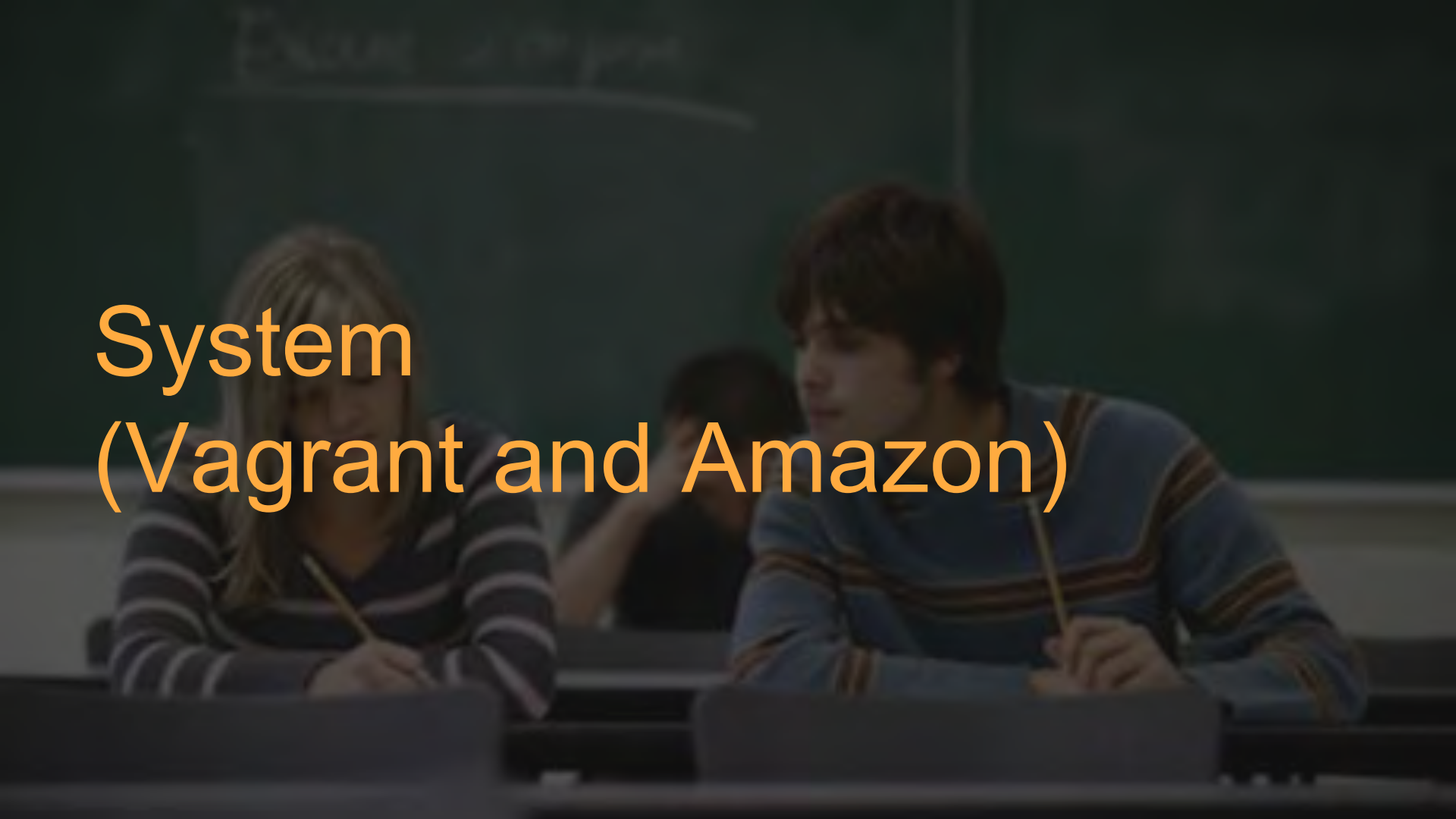

3. The Job


We Split the Job

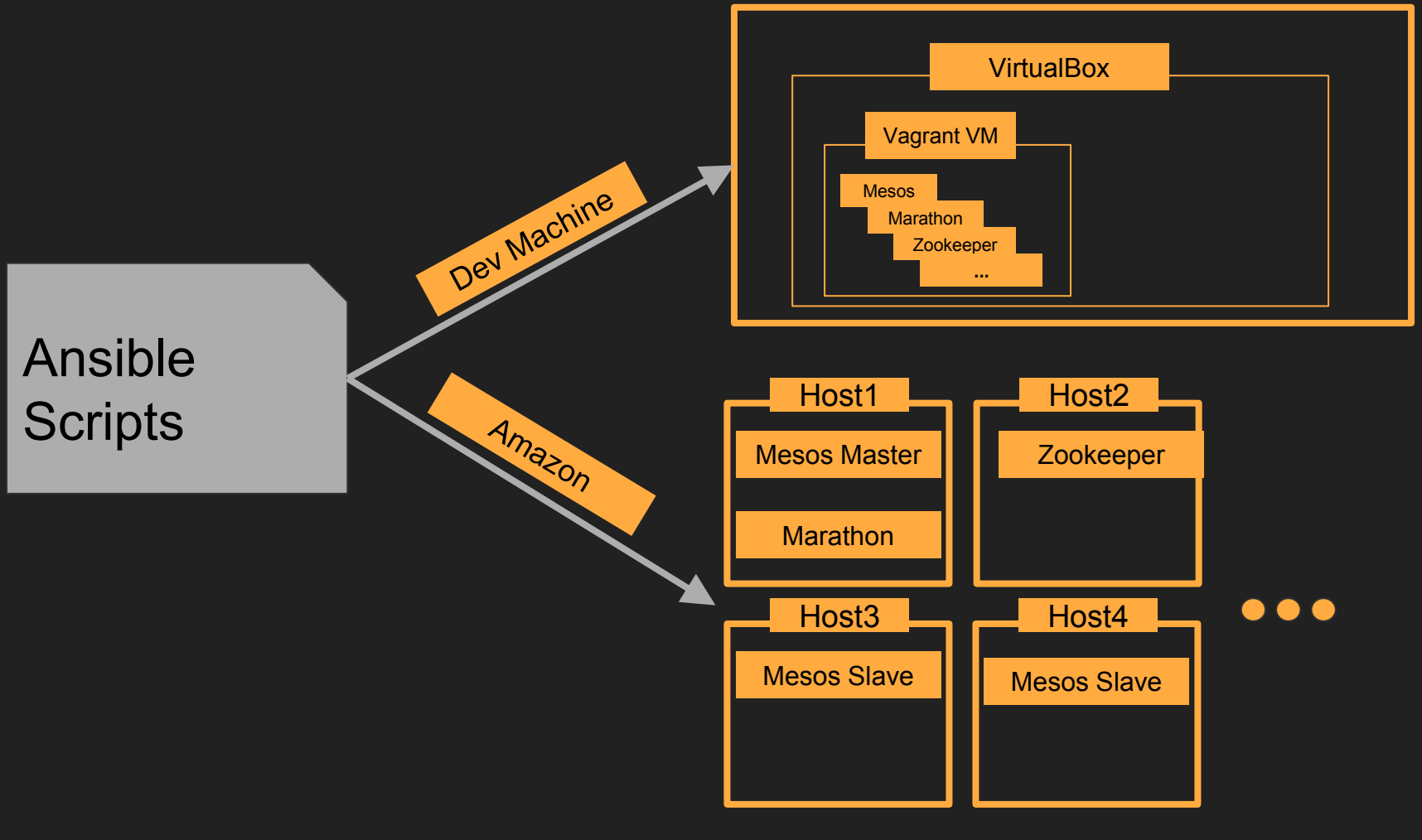
Splitting the Work

1. System (Vagrant and Amazon)
2. Sample Web Service and RPC Service
3. Deployment via CLI
4. RPC Service Discovery
5. Web Service Discovery

1. System (Vagrant and Amazon)
2. Sample Web Service and RPC Service
3. Deployment via CLI
4. RPC Service Discovery
5. Web Service Discovery



System (Vagrant and Amazon)


Ansible Scripts
Dev Machine
Amazon
VirtualBox
Vagrant VM
Mesos
Marathon
Zookeeper
...
Host1
Mesos Master
Marathon
Host2
Zookeeper
Host3
Mesos Slave
Host4
Mesos Slave

1. System (Vagrant and Amazon)

2. Sample Web Service and RPC Service

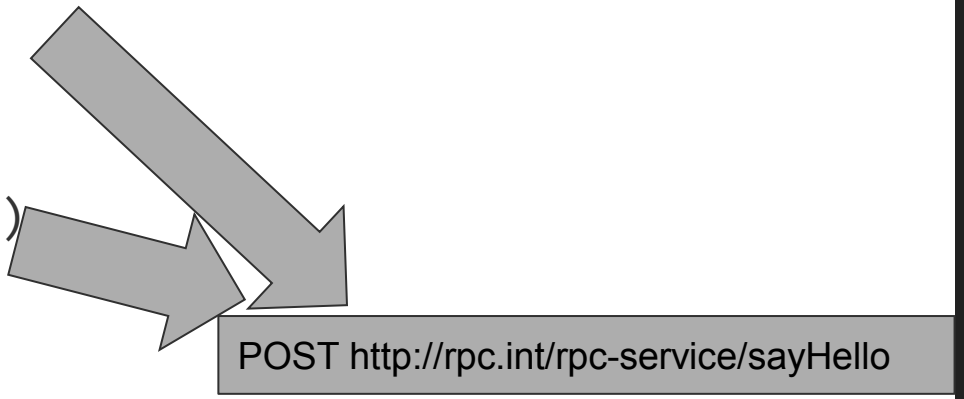
3. Deployment via CLI

4. RPC Service Discovery

5. Web Service Discovery

Example RPC Call

```
val helloServiceRpcClient = rpcClientFactory
    .rpcProxyFactory.builderFor(classOf[HelloService])
    .withStaticURL(new Url(s"http://${sys.env("RPC_INT")}")
    .withPart("rpc-service")
    .build()

helloServiceRpcClient.sayHello()
```



Splitting the Work

1. System (Vagrant and Amazon)
2. Sample Web Service and RPC Service
3. Deployment via CLI
4. RPC Service Discovery
5. Web Service Discovery

Remember?

```
$ npm install -g apollo

$ apollo artifact-deploy web-service:1.5.0 -n 20

$ curl web-service.demo-domain.com/....

$ apollo artifact-deploy web-service:1.6.0 -n 30
```

Deploying Instances To Marathon

```
const appId =

`/production/backend/${serviceName}`;

const appDeployResponse =
  marathon.client('PUT',
           `/apps${appId}`,
appConfig);
```

```
const appConfig = {
   id: appId,
   container: {
     type: 'DOCKER',
     docker: {
       image: `${serviceName)}:${version}`,
     }
   },
   env: {RPC_INT: 'localhost:9090'},
   labels: {
     microService: true,
     serviceName: serviceName
   },
   healthChecks: [{
     "path": '/health/is_alive'
   }]
   "cpus": 0.2,
   "mem": 200,
   "instances": parseInt(numberOfInstances)
};
```

Marathon...

- Sees how many instances needs to be deployed
- Uses Mesos to deploy the correct number of them on the hosts using Docker
- Waits for them to be alive using the health check
- The configuration is not a one-off
  - Marathon will always strive for it
  - If something dies (a host or an instance),

Marathon...

- Sees how many instances needs to be deployed
- Uses Mesos to deploy the correct number of them on the hosts using Docker
- Waits for them to be alive using the health check
- The configuration is not a one-off
  - Marathon will always strive for it
  - If something dies (a host or an instance),

Marathon...

- Sees how many instances needs to be deployed
- Uses Mesos to deploy the correct number of them on the hosts using Docker
- Waits for them to be alive using the health check
- The configuration is not a one-off
  - Marathon will always strive for it
  - If something dies (a host or an instance),

Marathon...

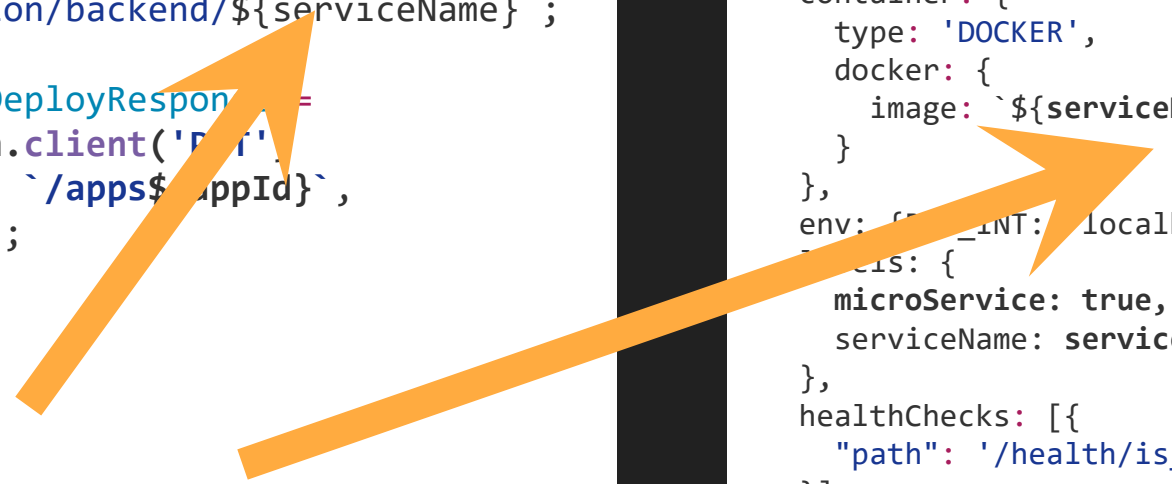
- Sees how many instances needs to be deployed
- Uses Mesos to deploy the correct number of them on the hosts using Docker
- Waits for them to be alive using the health check
- The configuration is not a one-off
  - Marathon will always strive for it
  - If something dies (a host or an instance),

What About Upgrading?

```
$ npm install -g apollo

$ apollo artifact-deploy web-service:1.5.0 -n 20

$ curl web-service.demo-domain.com/....

$ apollo artifact-deploy web-service:1.6.0 -n 30
```

Upgrading Instances

```
const appId =

`/production/backend/${serviceName}`;

const appDeployRespon  =
  marathon.client('D  r'
          `/apps$ appId}`,
appConfig);
```



```
const appConfig = {
   id: appId,
   container: {
     type: 'DOCKER',
     docker: {
       image: `${serviceName)}:${version}`,
     }
   },
   env: {  _INT: 'localhost:9090'},
   l  els: {
     microService: true,
     serviceName: serviceName
   },
   healthChecks: [{
     "path": '/health/is_alive'
   }]
   "cpus": 0.2,
   "mem": 200,
   "instances": parseInt(numberOfInstances)
 };
```

Marathon...

- Notices the configuration changed
- Strives for it to be true
- Uses Blue/Green Deployment to ensure constant uptime

Marathon...

- Notices the configuration changed
- Strives for it to be true
- Uses Blue/Green Deployment to ensure constant uptime

Marathon...

- Notices the configuration changed
- Strives for it to be true
- Uses Blue/Green Deployment to ensure constant uptime

The First Three Tasks

1. System (Vagrant and Amazon)
2. Sample Web Service and RPC Service
3. Deployment via CLI
4. RPC Service Discovery
5. Web Service Discovery

The First Three Tasks

1. System (Vagrant and Am
2. Sample Web Ser ice
3. Deploym
4. RPC ery
5. Web Se Discovery

We did it in **3** days!

And What About The Rest?

1. System (Vagrant and Amazon)
2. Sample Web Service and RPC Service
3. Deployment via CLI
4. RPC Service Discovery
5. Web Service Discovery

4. The Change of Plan

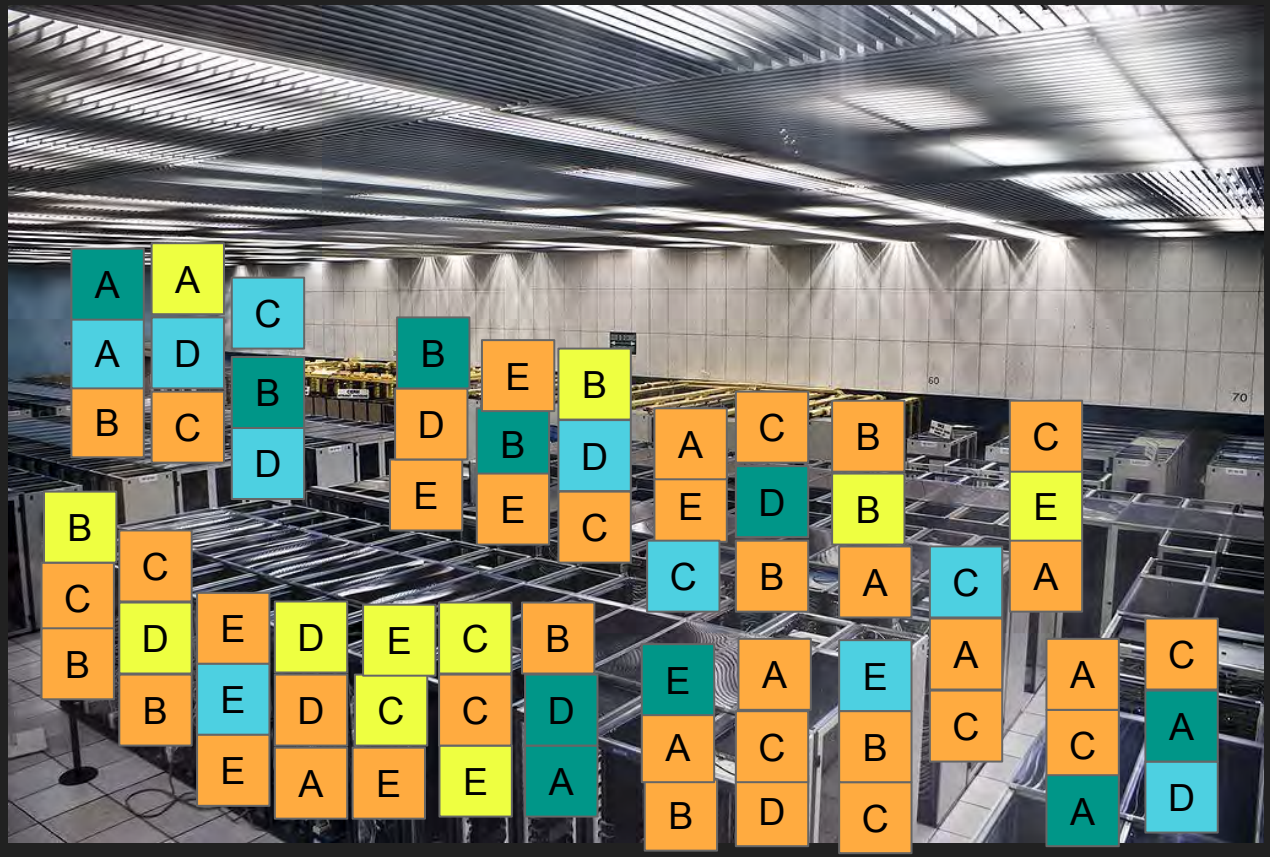
3rd Party Service Discovery!

So What Else?


Staging!

Production is just another **environment**

Environments

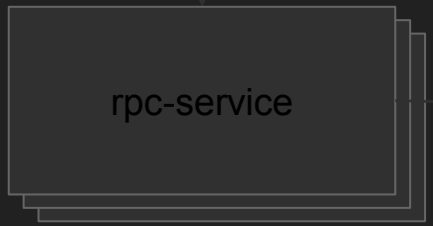


How does this look inside the CLI?

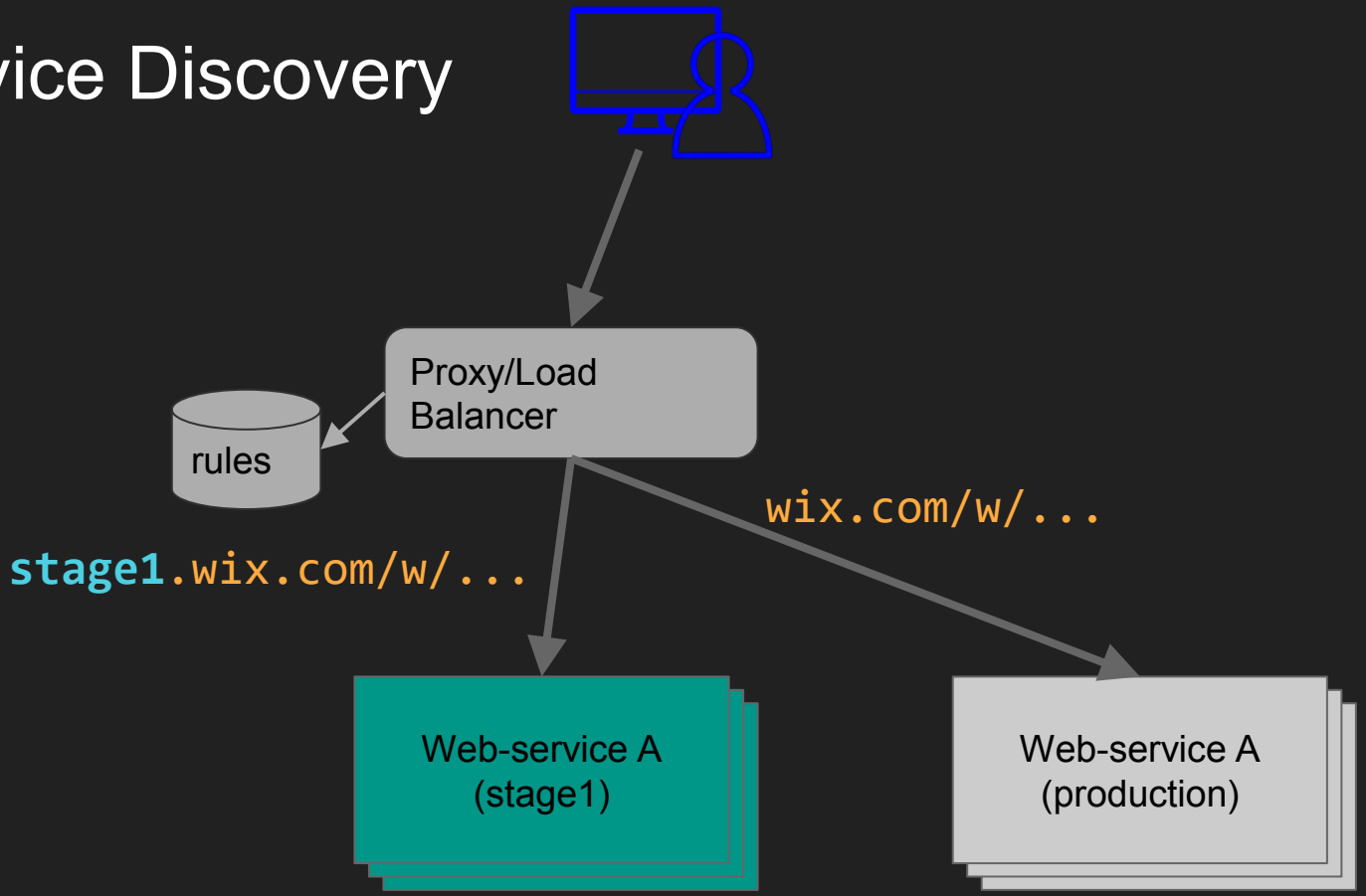
```
$ npm install -g apollo

$ apollo artifact-deploy web-service:1.5.0
                                    -e stage1 -n 20

$ apollo artifact-deploy web-service:1.6.0 -n 1
                                        --testbed

$ apollo artifact-deploy web-service:1.6.0 -n 30
```

And from the browser?

`stage1.wix.com/w/...`

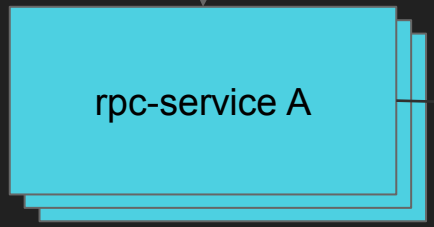


Web Service Discovery



And RPC Services?

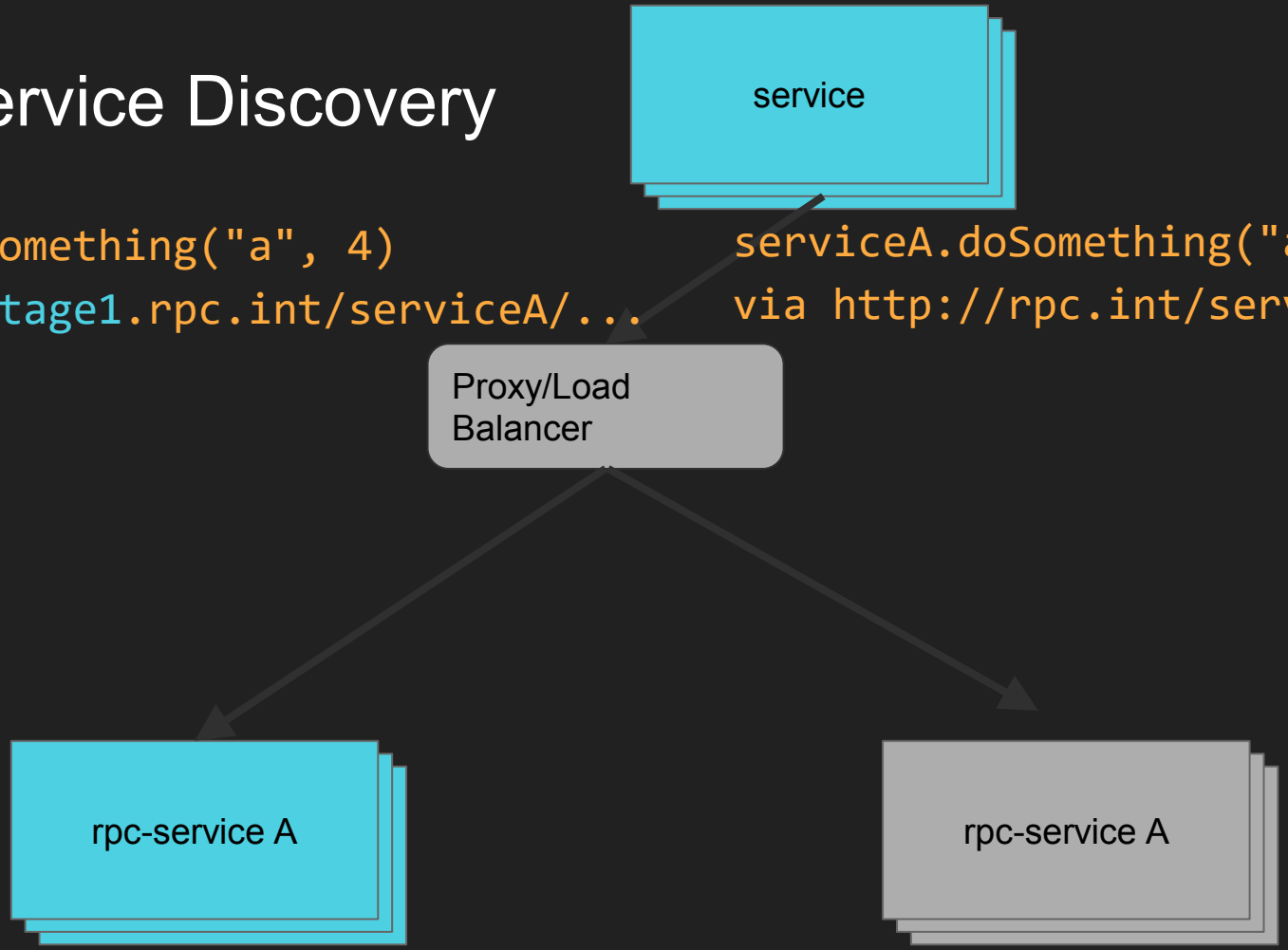

```
serviceA.doSomething("a", 4)
```



RPC Service Discovery

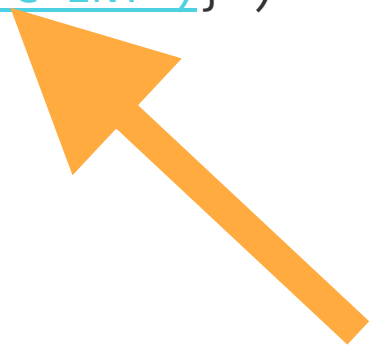
```
serviceA.doSomething("a", 4)
via http://stage1.rpc.int/serviceA/...
```

```
serviceA.doSomething("a", 4)
via http://rpc.int/serviceA/...
```

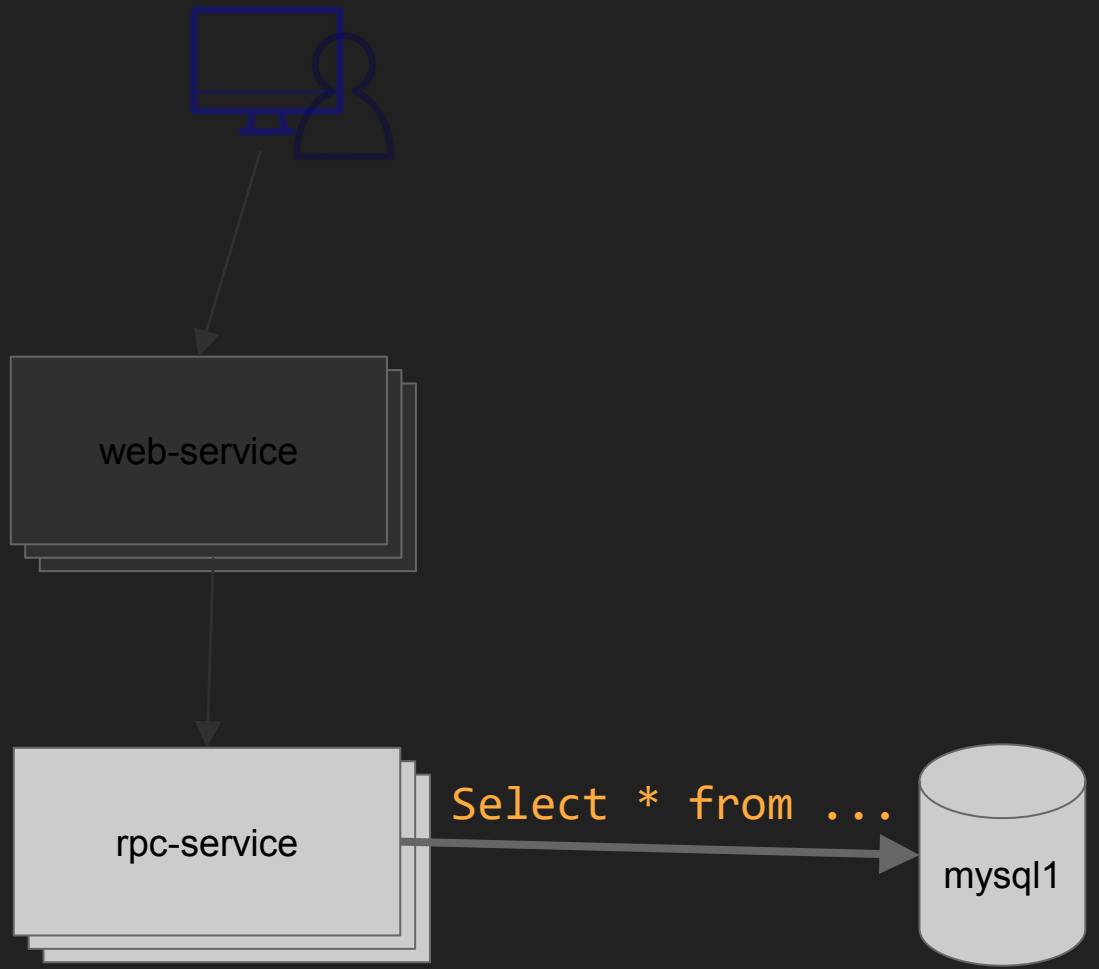


Example RPC Call

```
val helloServiceRpcClient = rpcClientFactory
    .rpcProxyFactory.builderFor(classOf[HelloService])
    .withStaticURL(
              new Url(s"http://${sys.env("RPC_INT")}")
    .withPart("rpc-service")
    .build()

helloServiceRpcClient.sayHello()
```

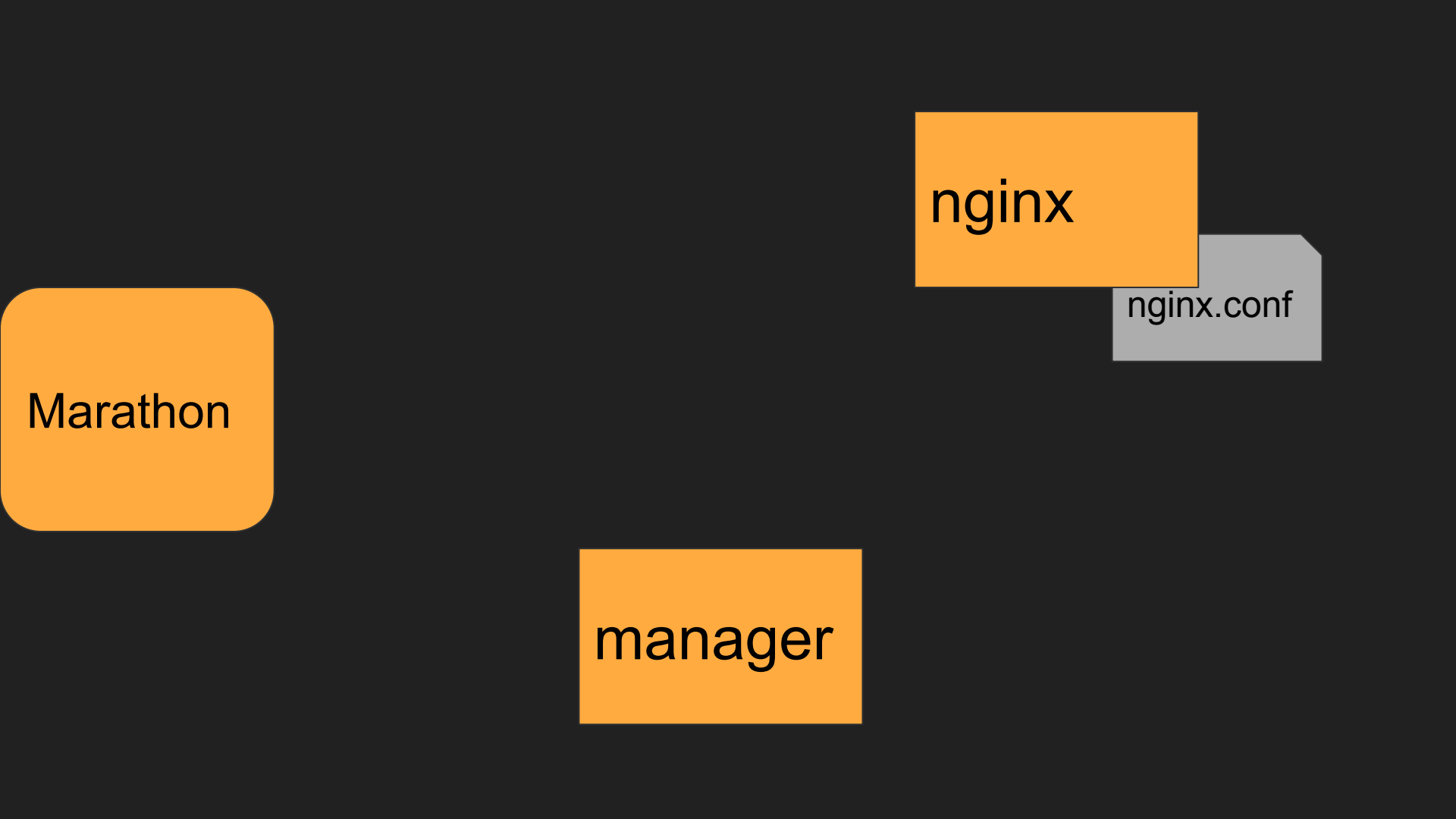


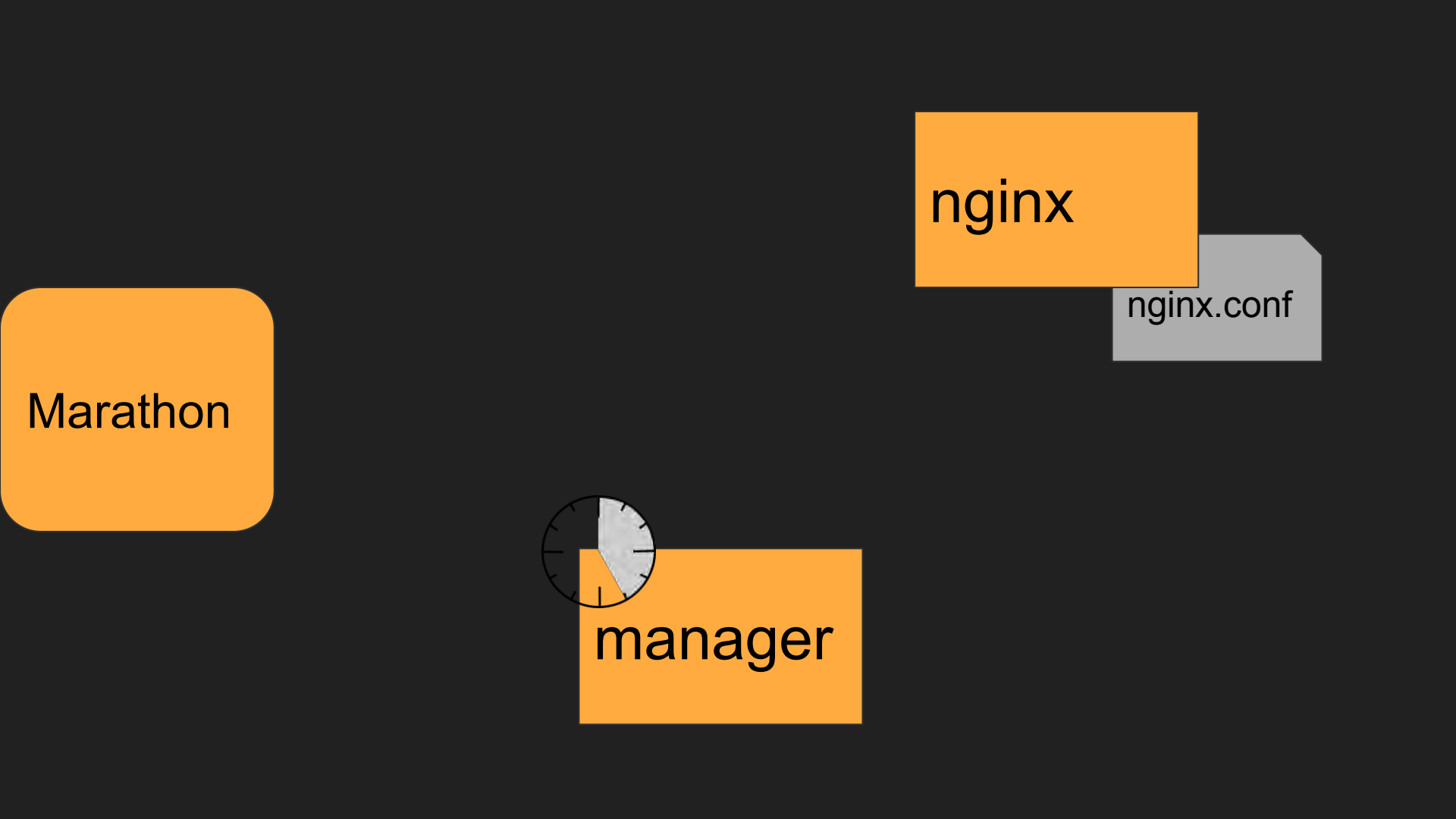
And 3rd Party Services?



5. It All Comes Together


Same Difference

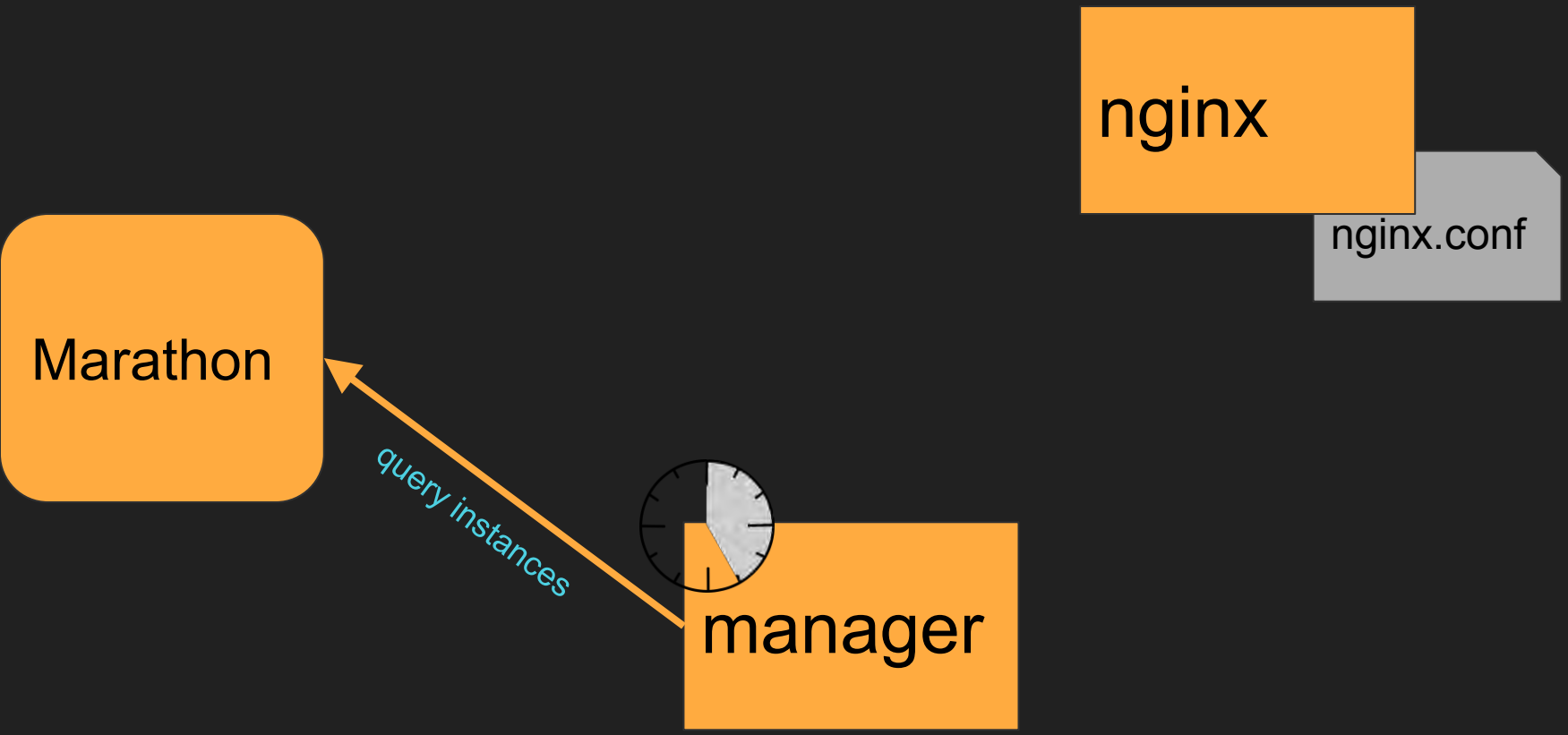

nginx
nginx.conf
Marathon
manager

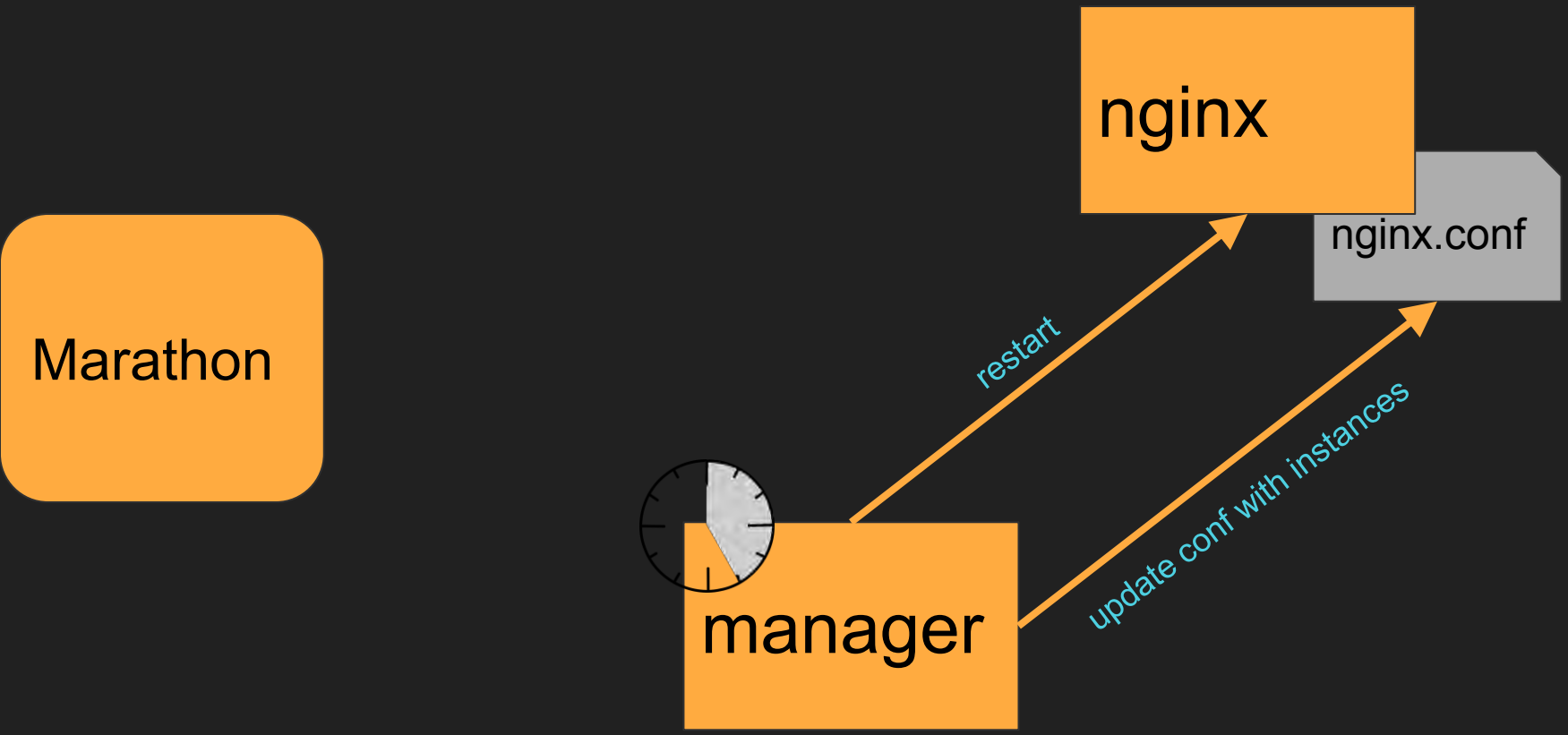

nginx
nginx.conf
Marathon
manager

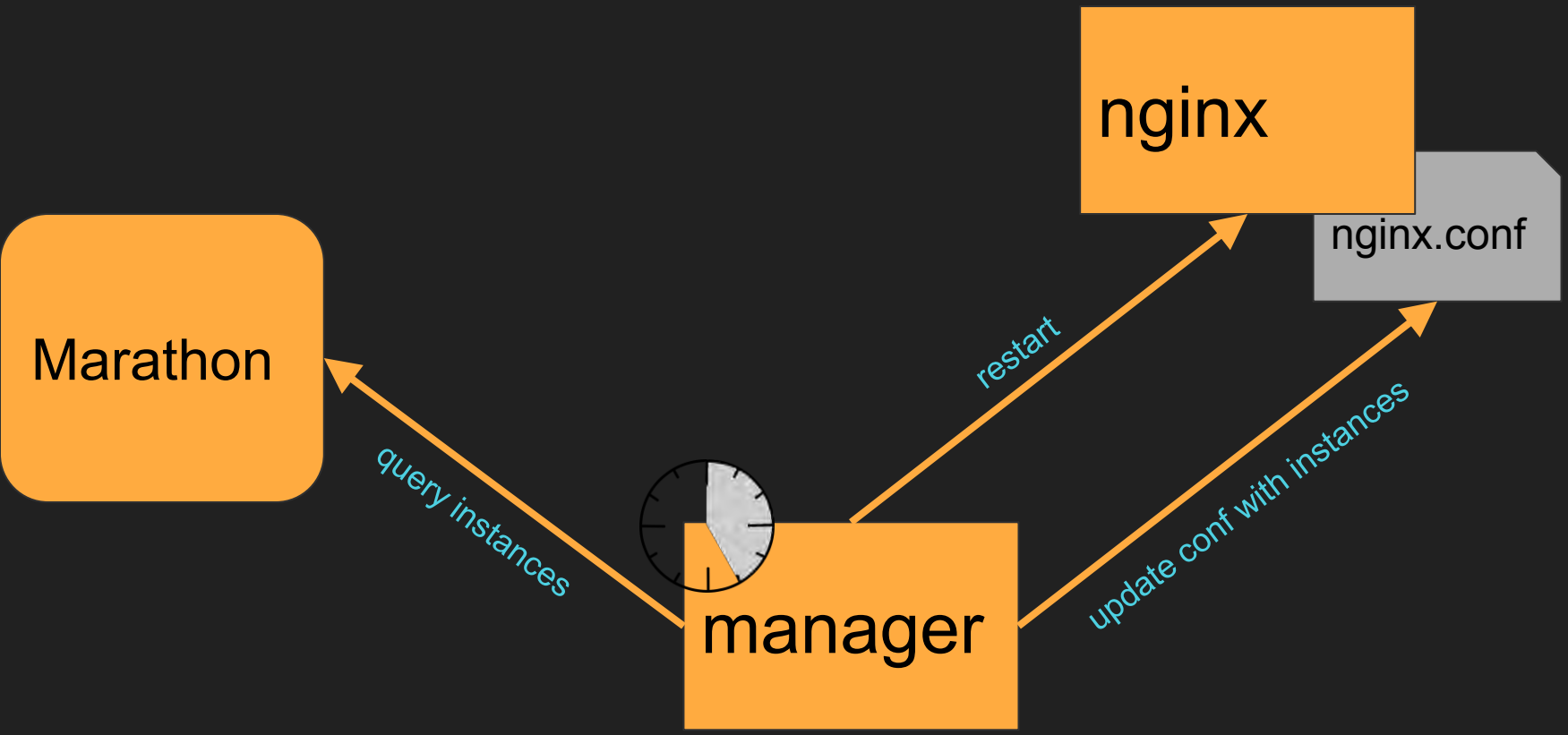
Remember?

```
const appId =

`/production/backend/${serviceName}`;

const appDeployResponse =
  marathon.client('PUT',
          `/apps${appId}`,
appConfig);
```

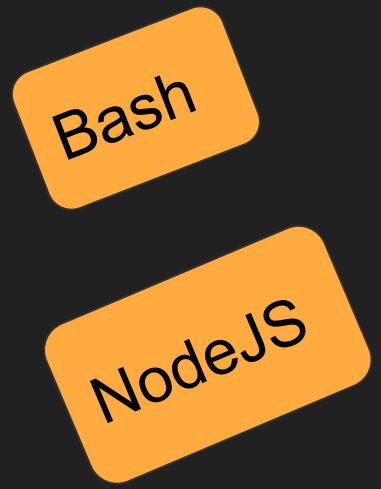
```
const appConfig = {
   id: appId,
   container: {
     type: 'DOCKER',
     docker: {
       image: ${serviceName)}:${version}`,
     }
   },
   env: {RPC_INT: ‘localhost:9090’, ...},
   labels: {
     microService: true,
     serviceName: serviceName
   },
   healthChecks: [{
     "path": '/health/is_alive'
   }]
   "cpus": 0.2,
   "mem": 200,
   "instances": parseInt(numberOfInstances)
 };
```


nginx
nginx.conf
Marathon
query instances
manager


Marathon
nginx
nginx.conf
restart
update conf with instances
manager


nginx
nginx.conf
Marathon
restart
query instances
update conf with instances
manager

Language?



Bash is easier

Generating nginx.conf from information in Marathon

```
request_url="http://$marathon_host/v2/apps?embed=app.tasks"
json=$(curl --fail --silent $request_url)
environments=$(echo $json | \
      jq -r '[.apps[] | .labels.environment] | unique)

for environment in $environments; do
    export environment
    apps=$(echo $json | jq -r '.apps[] | \
          select(.labels.environment == env.environment)')
    app_ids=$(echo $apps | jq -r '.id' | \
                    sed 's/-testbed//g' | sort | uniq)

    for app_id in $app_ids; do
       ....
    done
done
```

```
app=$(echo $apps | jq -r 'select(.id == env.app_id),
select(.id == env.app_id + "-testbed")')

artifactId=$(echo $app | jq -r '.labels.artifactId' | uniq)
servers=$(echo $app | jq -r '.tasks[] |
select(.healthCheckResults[0].alive == true) | .host + ":" +
(.ports[0] | tostring)')

# Generate the part of the nginx.conf for each server where
this app is in
```

The First Three Tasks

1. System (Vagrant and Am
2. Sample Web Ser ice
3. Deploym
4. RPC ery
5. Web Se Discovery

Another **3** Days!

Let’s Recap

apollo deploy A -n 2 -e s

apollo deploy B -n 2 -e s

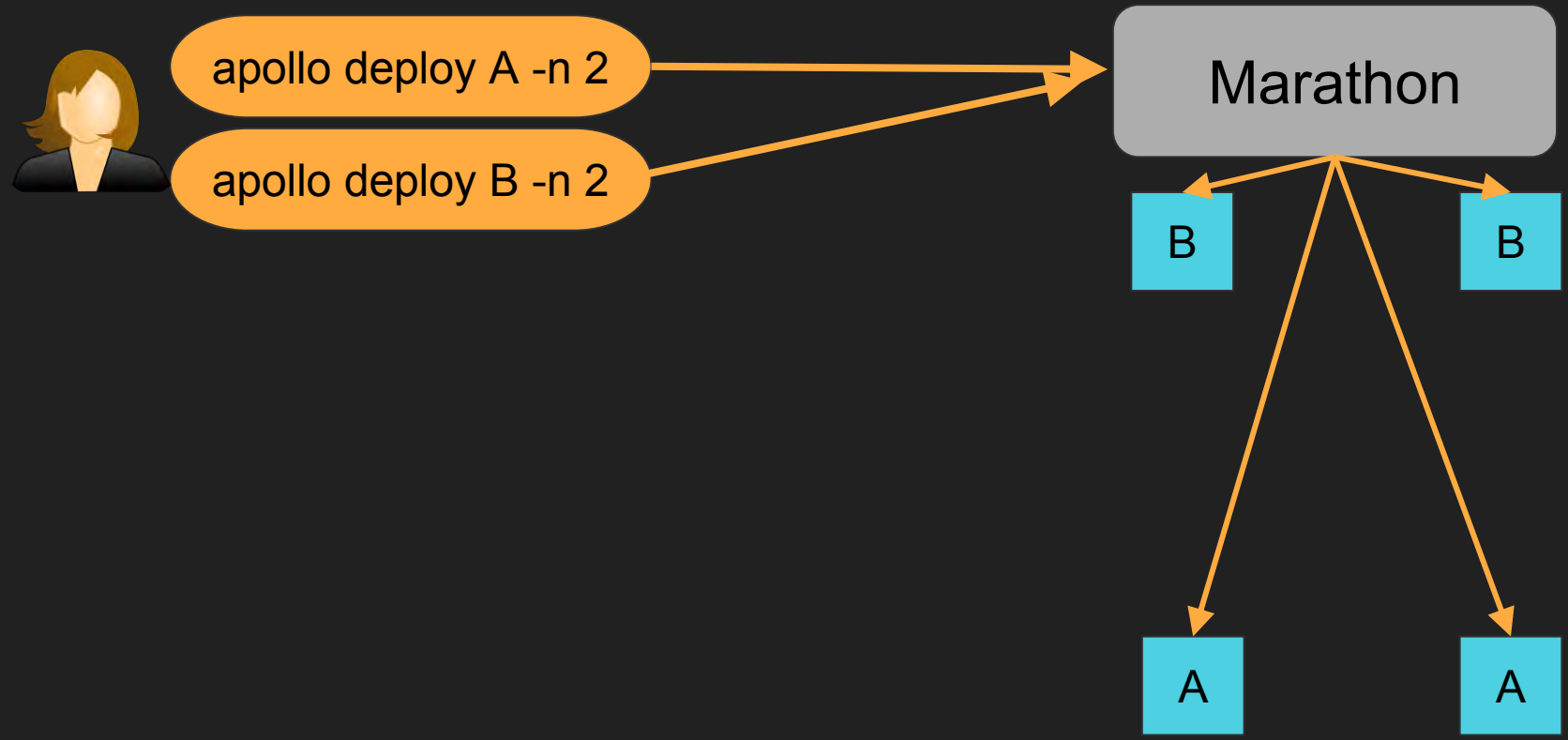
apollo deploy A -n 2 -e s

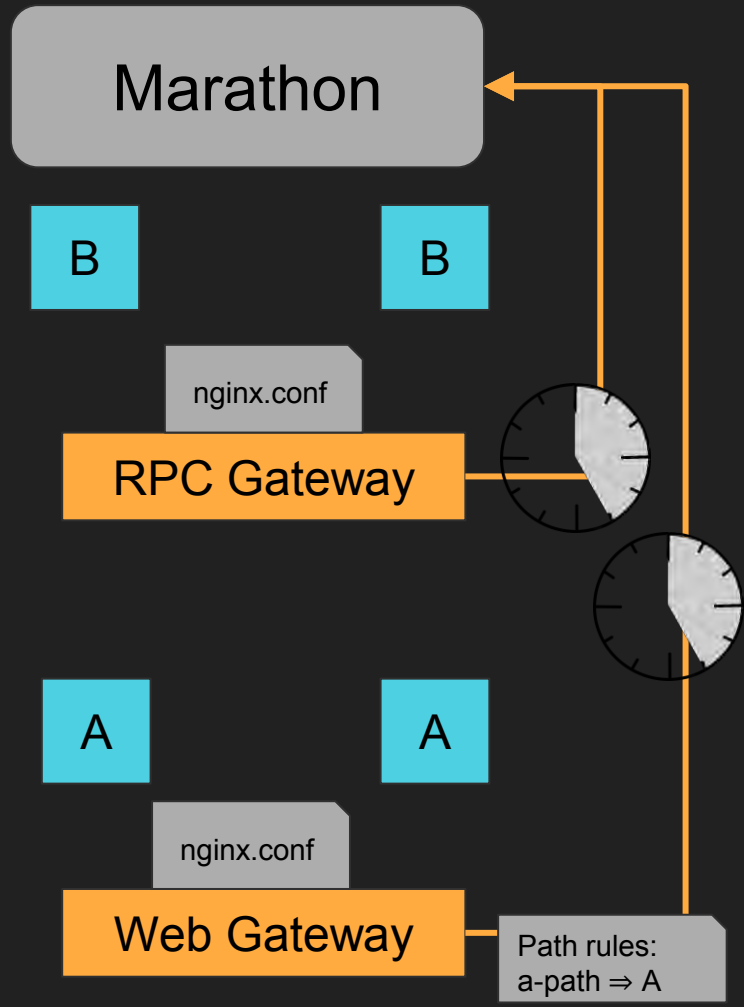
apollo deploy B -n 2 -e s

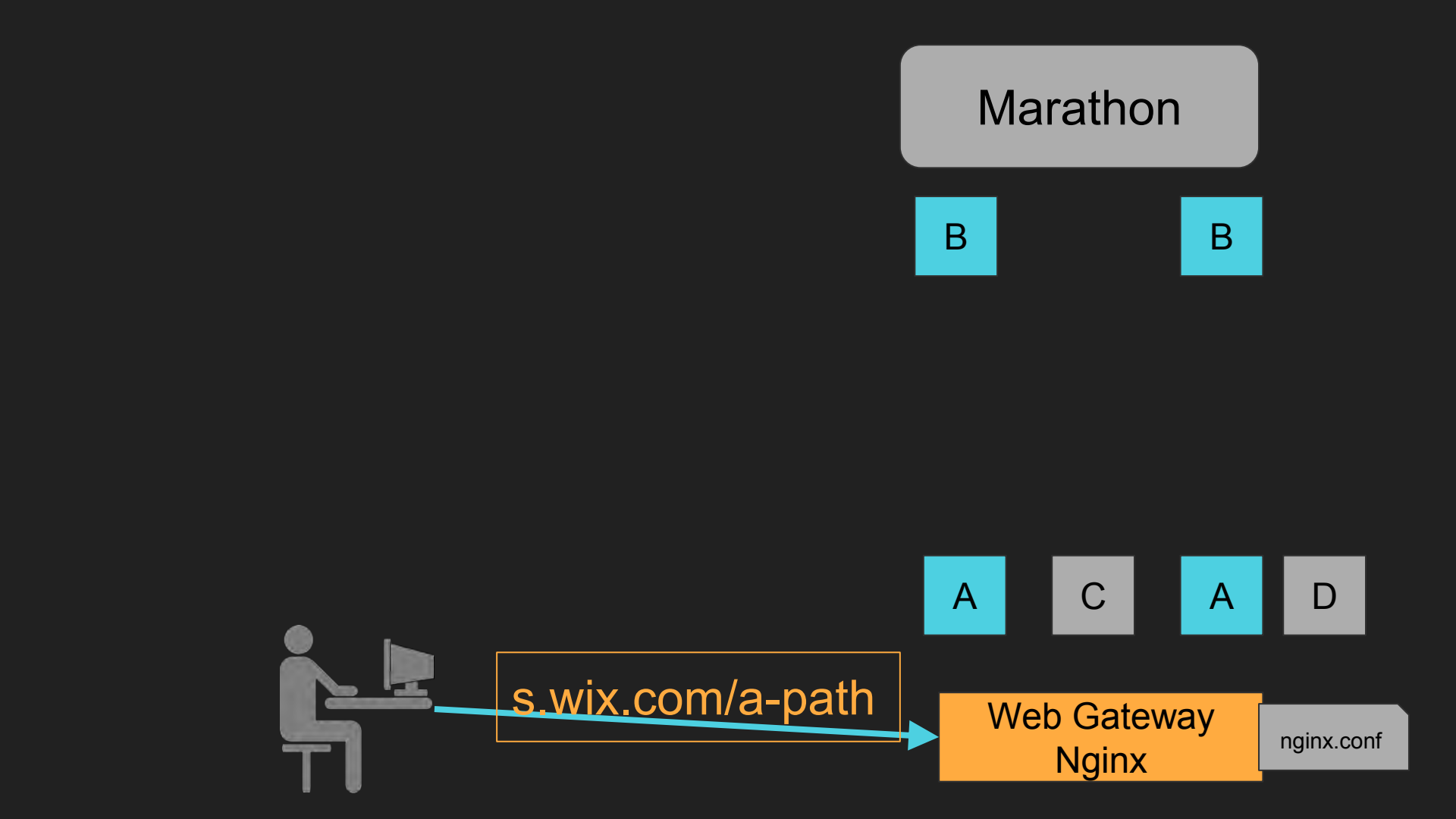
Marathon

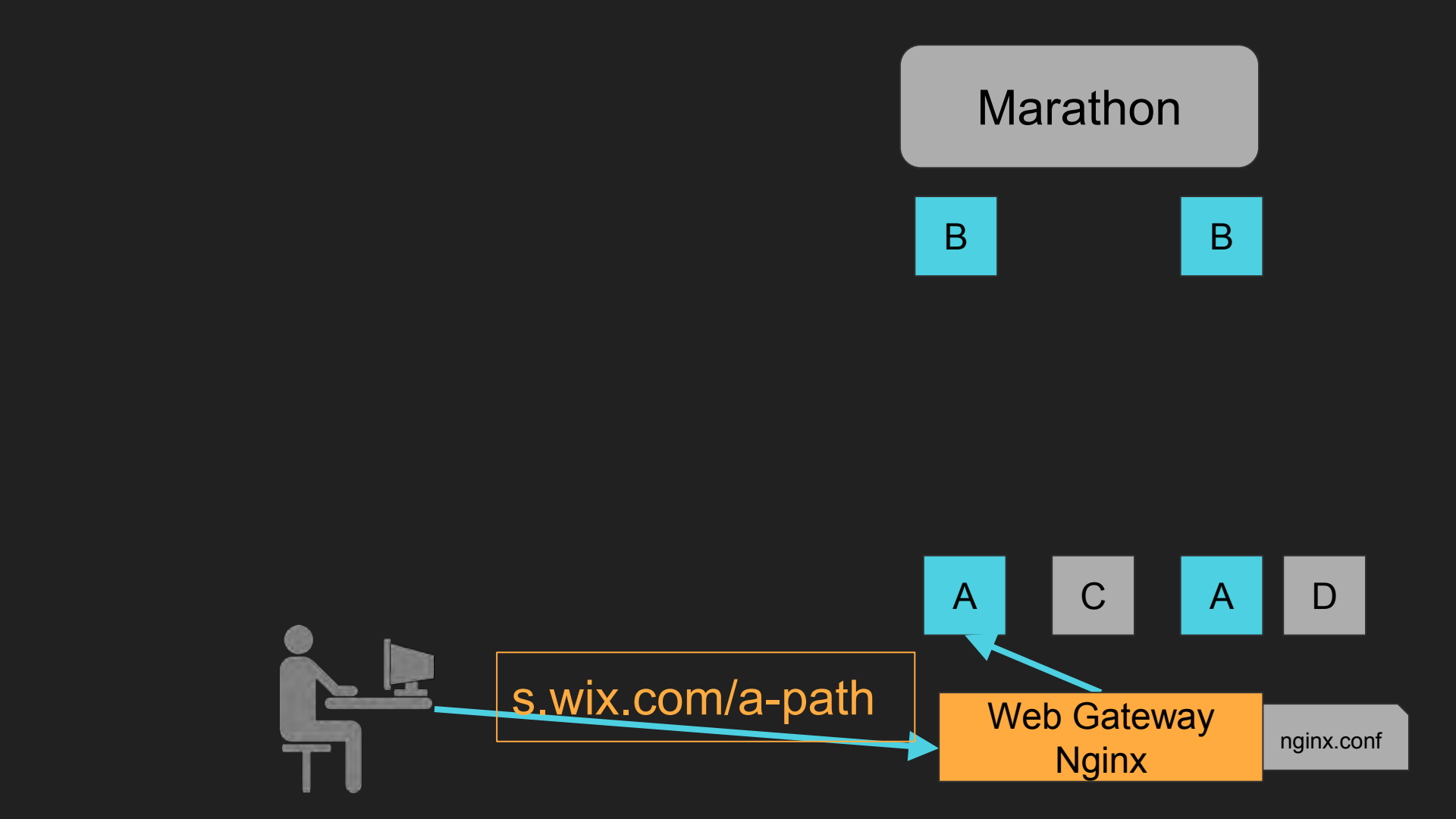
POST json configuration

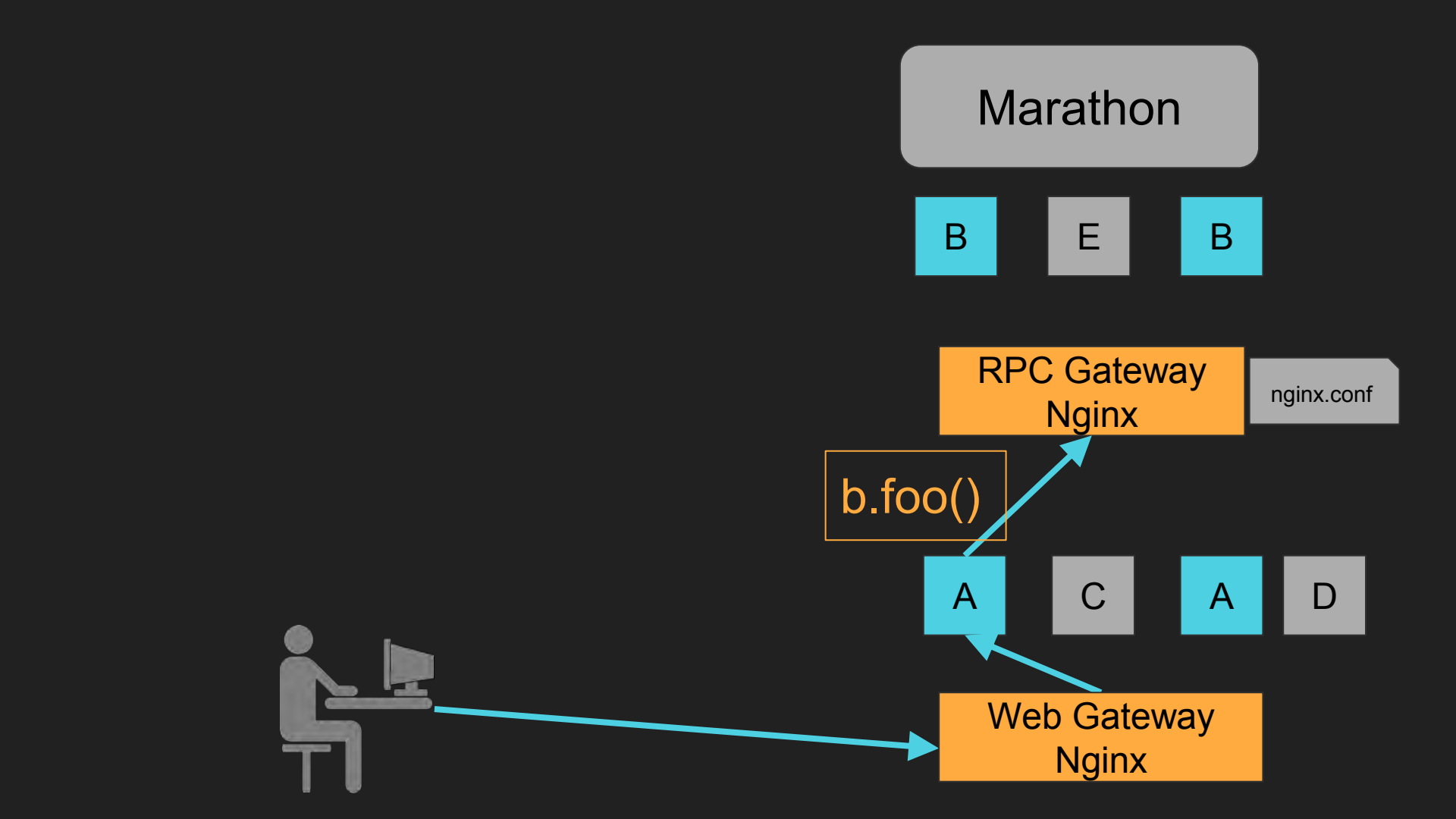
- # of instances
- RPC_INT pointing to s.rpc.int.
- Docker image to use

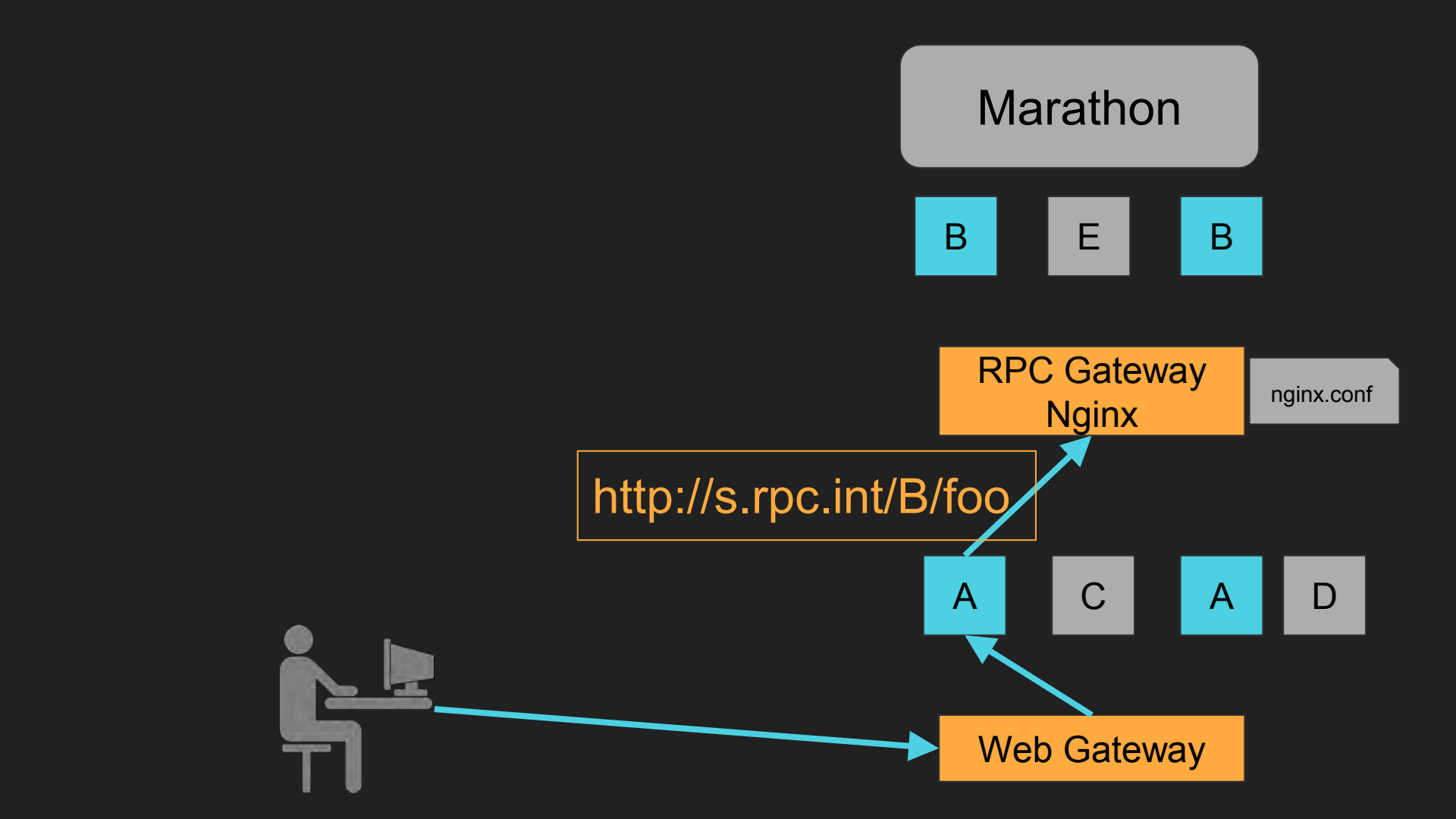

apollo deploy A -n 2
apollo deploy B -n 2
Marathon
B
B
A
A

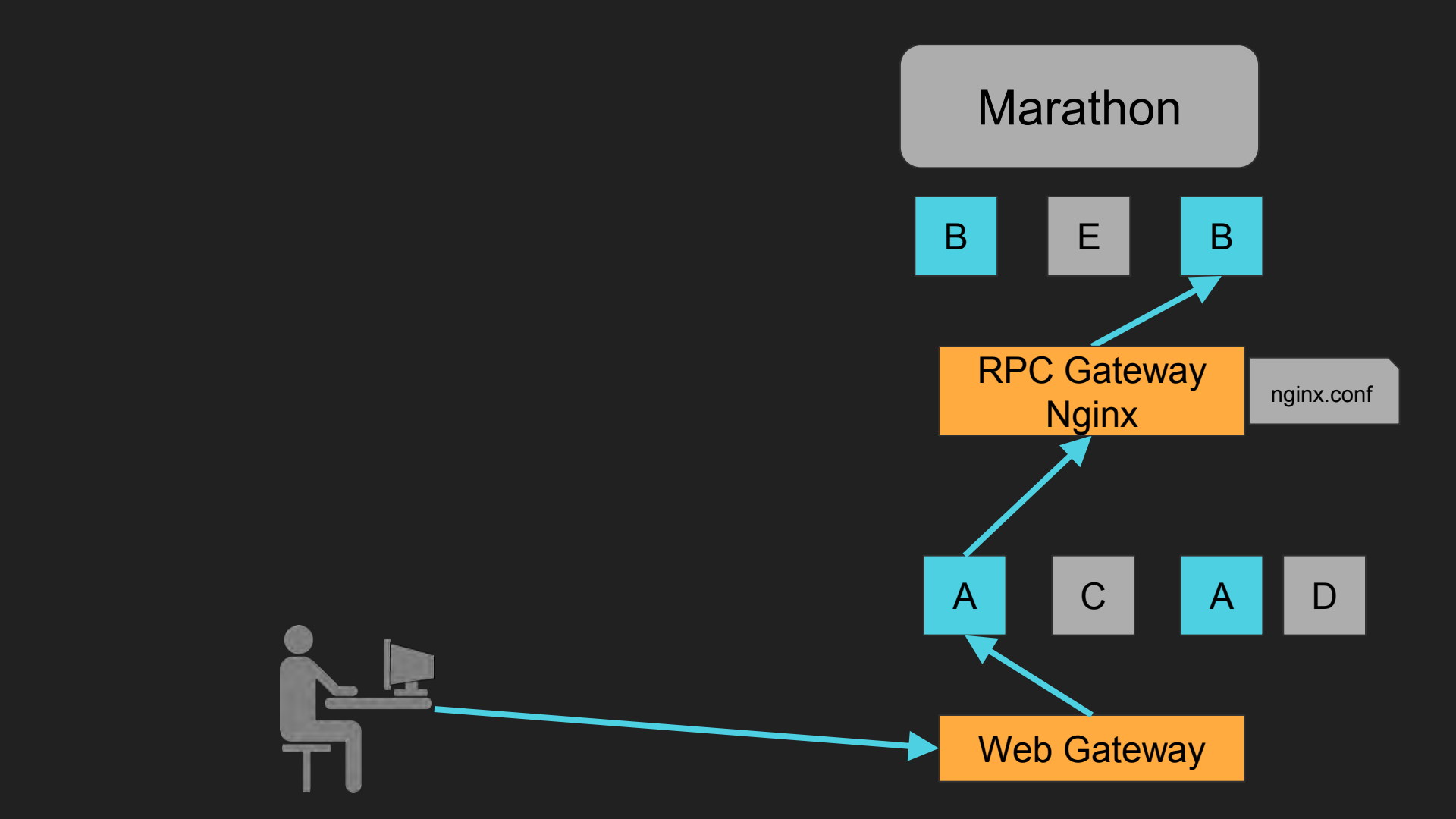

Marathon
B
B
nginx.conf
RPC Gateway
A
A
nginx.conf
Web Gateway
Path rules:
a-path ⇒ A

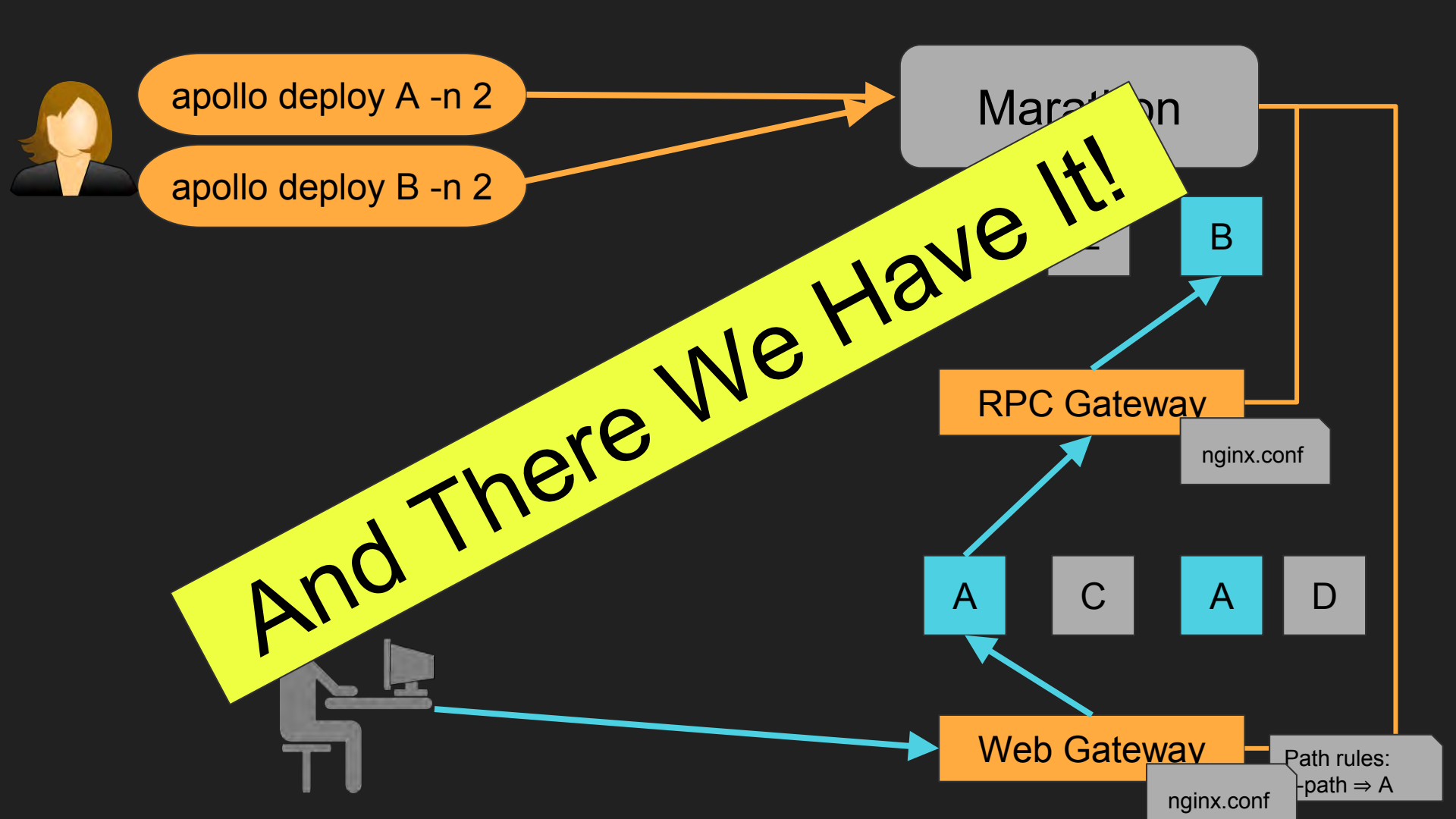

Marathon
B
B
A
C
A
D
s.wix.com/a-path
Web Gateway
Nginx
nginx.conf

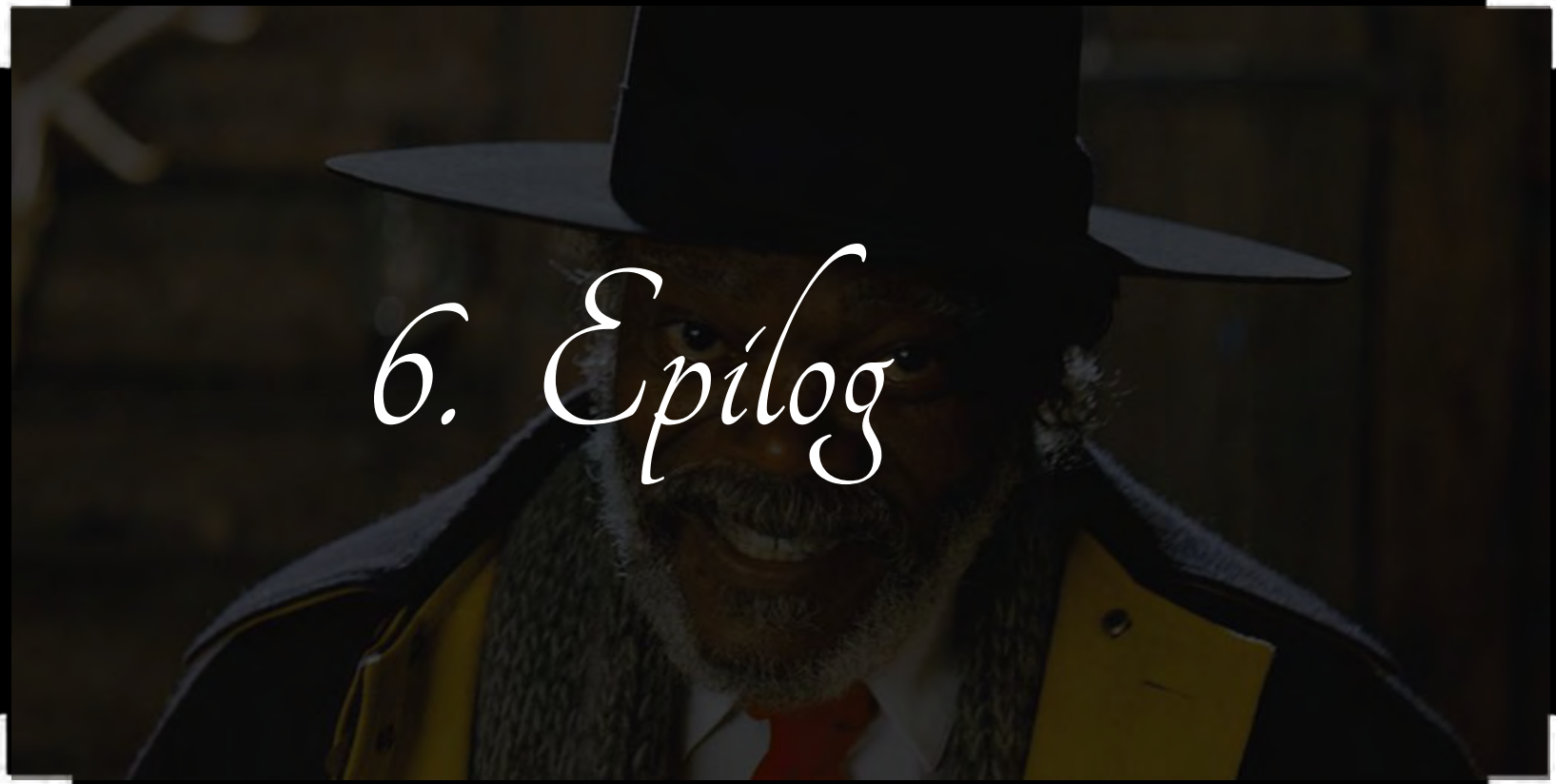

Marathon
B
B
A
C
A
D
s.wix.com/a-path
Web Gateway
Nginx
nginx.conf


Marathon
B
E
B
RPC Gateway
Nginx
nginx.conf
b.foo()
A
C
A
D
Web Gateway
Nginx


Marathon
B
E
B
RPC Gateway
Nginx
nginx.conf
http://s.rpc.int/B/foo
A
C
A
D
Web Gateway


Marathon
B
E
B
RPC Gateway
Nginx
nginx.conf
A
C
A
D
Web Gateway


apollo deploy A -n 2
apollo deploy B -n 2
Marathon
B
RPC Gateway
nginx.conf
A
C
A
D
Web Gateway
Path rules:
-path ⇒ A
nginx.conf
And There We Have It!



6. Epilog

We won!

End Result

- Micro-services Infra, supporting…
- Blue-green deployment
- 3rd Party Services
- Staging environments
- Built in 6 days

And on the 7th Day?


We Rested


We’re Jewish, No?

Thank You

@giltayar