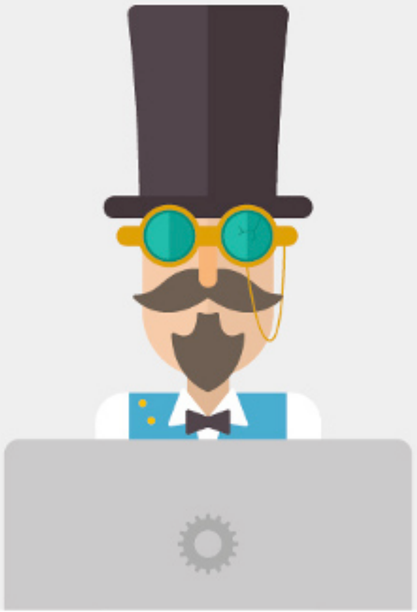
Capacity Testing with Docker

An Experience

Daniel Brown



geecon

About Me

Daniel Brown
@_dlpb

Engineer at eBay
Working in London
working on Scala, Akka, Spray projects

Arduino and Embedded Electronics for fun!
Monitor all the things

(Oh, and 19th Century detective novels)

Motivation

Integration tests that use dependencies you don’t control are slow.

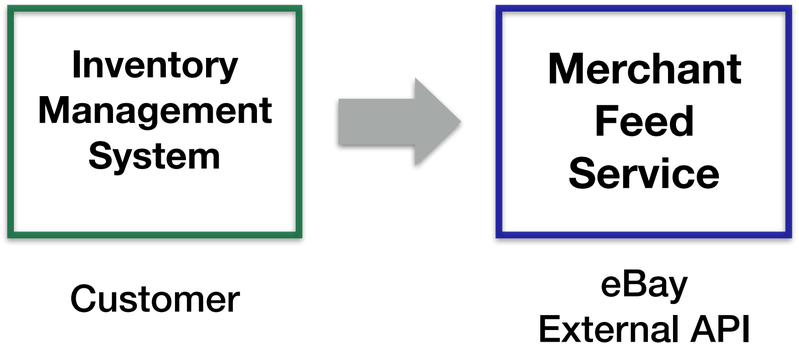
How can we reduce the time our integration tests take?

Motivation

Clever stub management with Docker

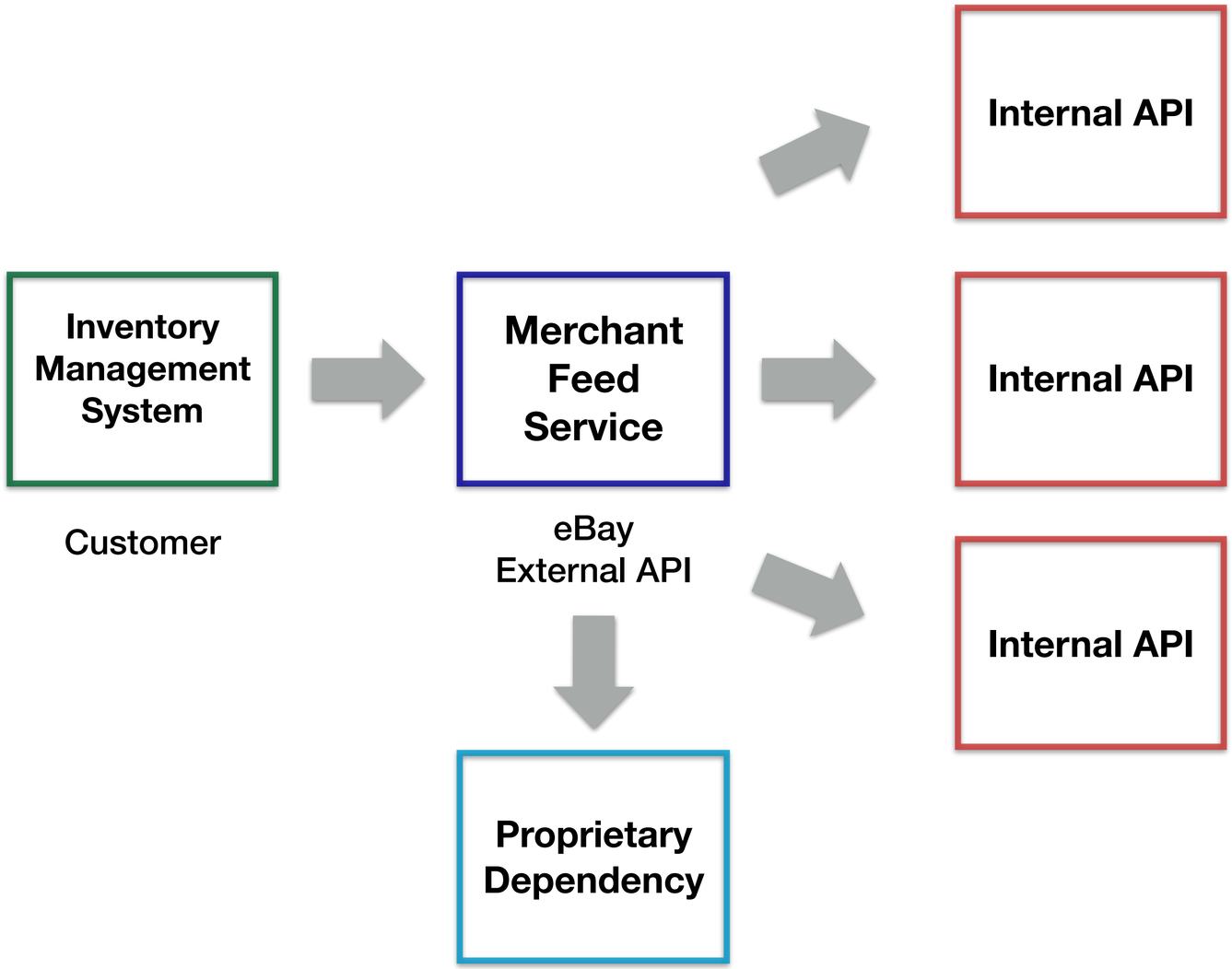
Background

Merchant Feed Service



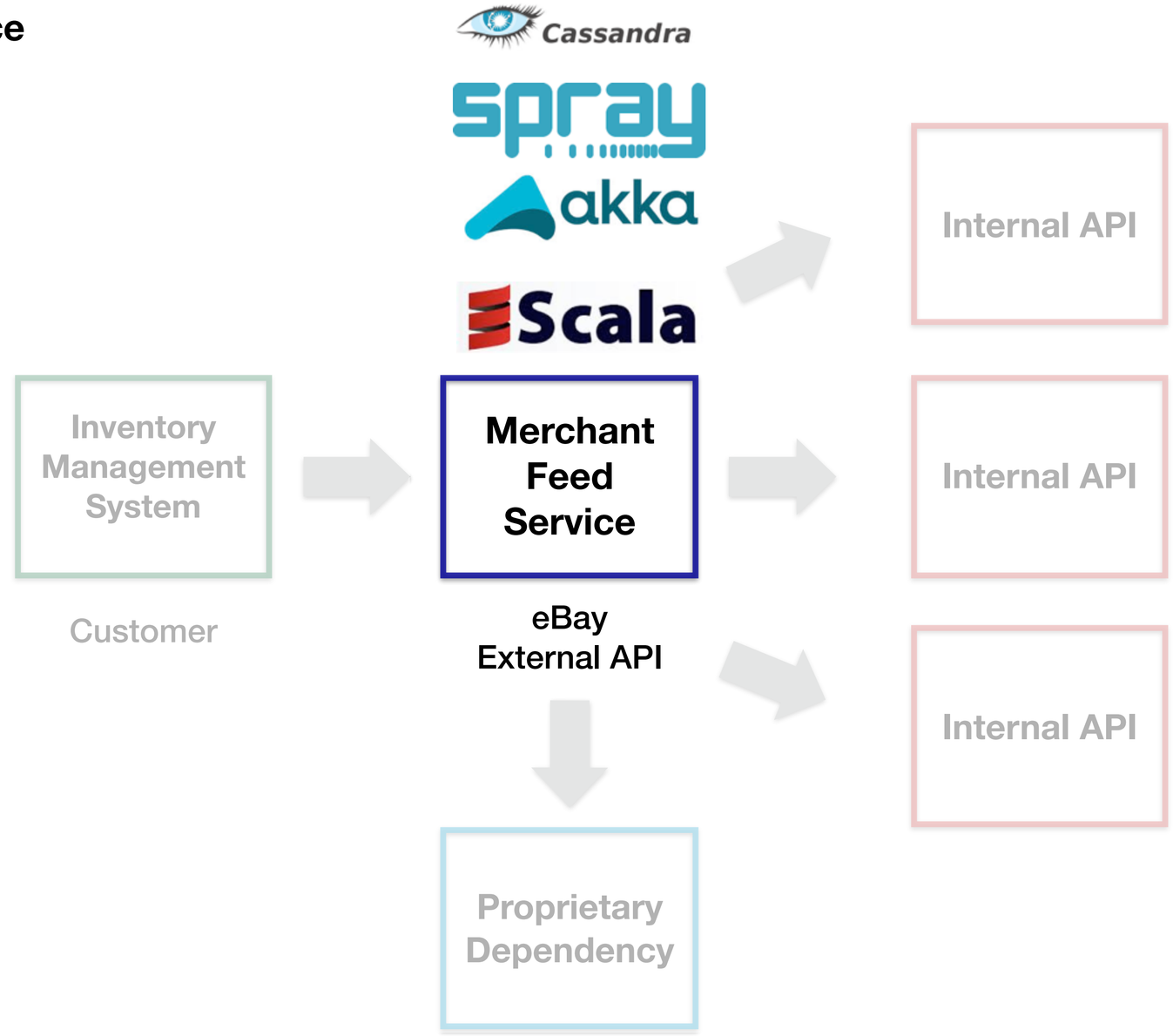
Background

Merchant Feed Service



Background

Merchant Feed Service



Background

Our Goal

Reduce the end-to-end test time

Merchant Feed Service

Testing Nightmares

Red Tests

Obtaining Data

Image: https://www.flickr.com/photos/x1brett/

Dependency Nightmares

Untested

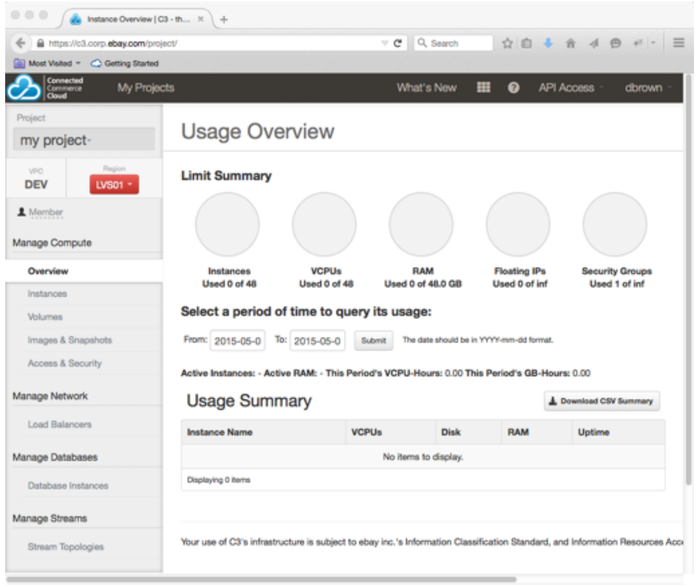
Unreliable

Slow

Image: https://www.flickr.com/photos/x1brett/

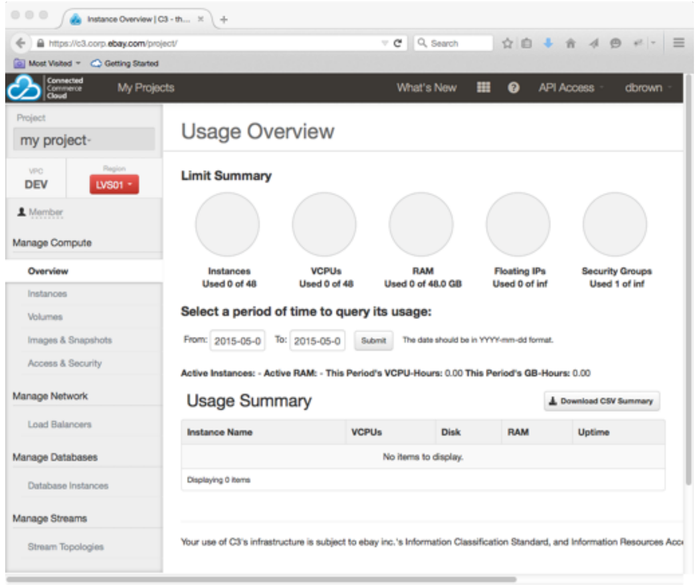
Virtual Machines

- eBay’s default way of working
- easy to provision (through a UI)
- In the “cloud”
- Choice of OS’s



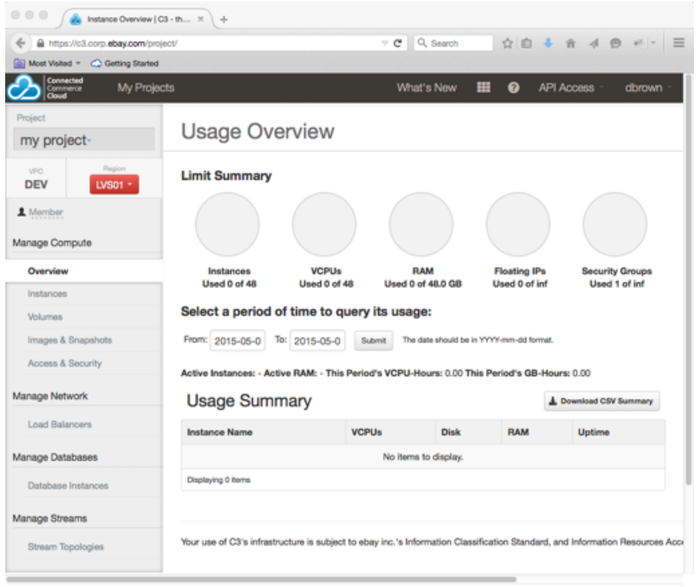
Virtual Machines

- Slow to provision internally
  - Both API and UI
- We are based in London
  - Infrastructure is in the US



Virtual Machines

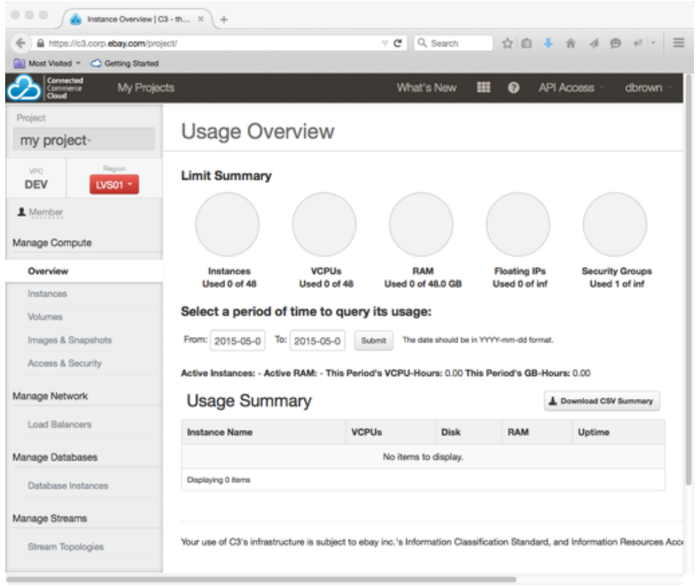
- Slow to provision internally
  - Both API and UI
- Machines can, and do, disappear
- Internal connection latency



Virtual Machines

Proprietary dependencies

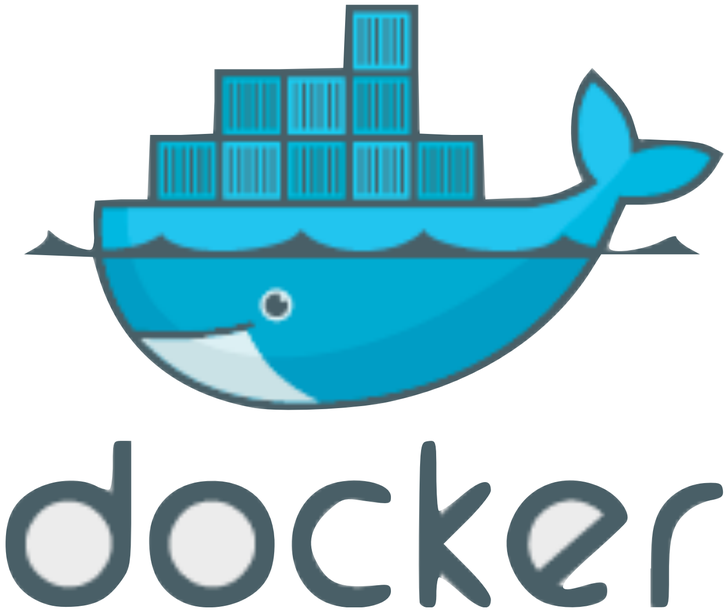
- Time consuming to set up
- Not possible to easily automate setup
  - e.g. db2, websphere
    (even with chef)



Enter Docker

Docker
noun

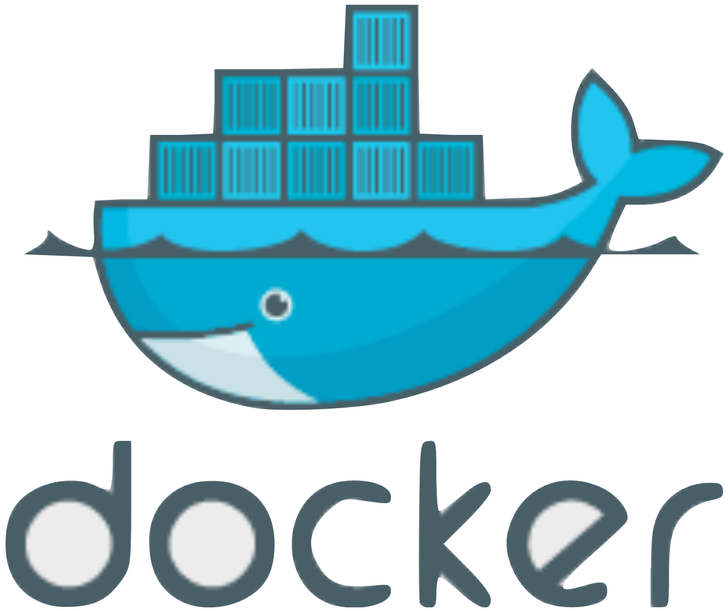
“An open platform for developers and sysadmins to build, ship and run applications



Enter Docker

Proprietary dependencies

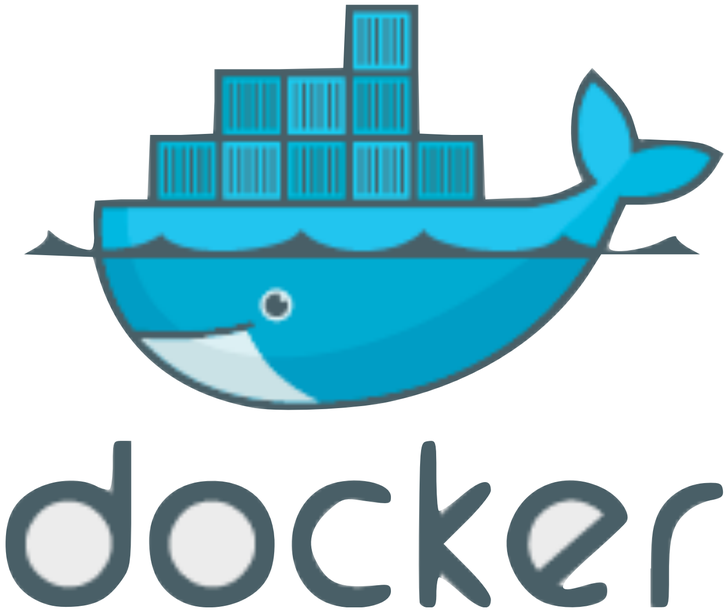
- Create images of dependencies



Enter Docker

Proprietary dependencies

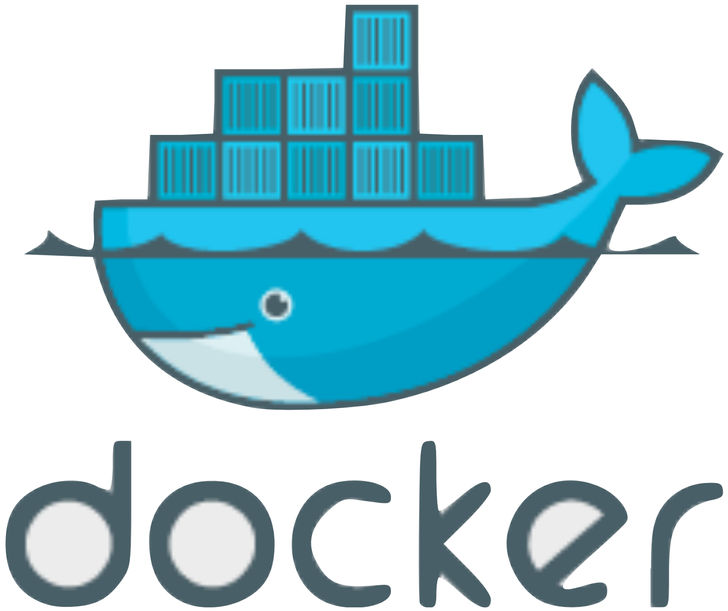
- Create images of dependencies
- Doesn't solve all setup issues
  - But only needs to be done once



Enter Docker

Proprietary dependencies

- Create images of dependencies
- Doesn't solve all setup issues
  - But only needs to be done once
- Less storage overhead than VM
  - db2+websphere = GBs of data

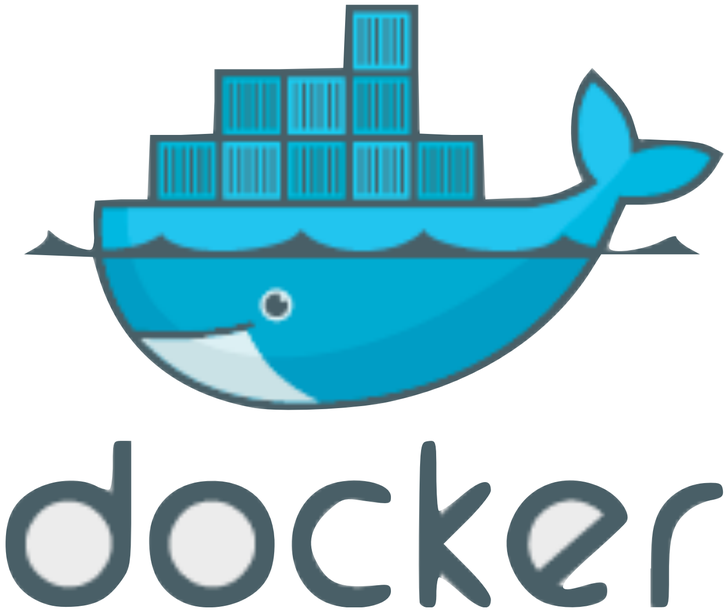


Enter Docker

Proprietary dependencies

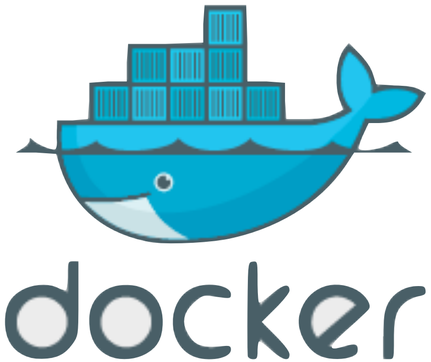
- Create images of dependencies
- Doesn't solve all setup issues
  - But only needs to be done once
- Less storage overhead than VM
  - db2+websphere = GBs of data

WIN!



Vanilla Docker

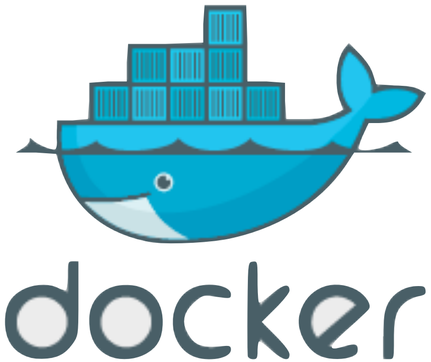
- Docker Images are good
- But, we don’t want to run them all manually




```
docker run -p 8081:8081 server:1.0.0



docker run --link server:server -p 80:80 frontend:1.0.0
```

Vanilla Docker

- Docker Images are good
- But, we don't want to run them all manually

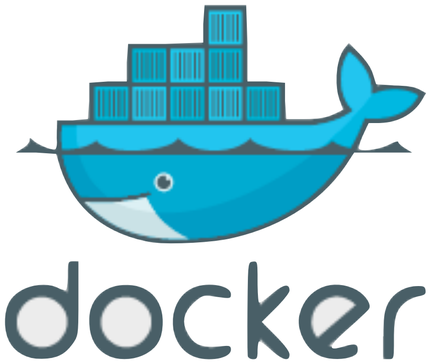



```
docker run -p 8081:8081 server:1.0.0

docker run --link server:server -p 80:80 frontend:1.0.0
```

Image Name & Version

Vanilla Docker

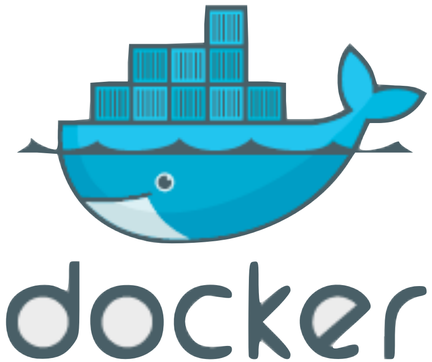
- Docker Images are good
- But, we don't want to run them all manually




```
docker run -p 8081:8081 server:1.0.0

                          Specify the ports

docker run --link server:server -p 80:80 frontend:1.0.0
```

Vanilla Docker

- Docker Images are good
- But, we don't want to run them all manually



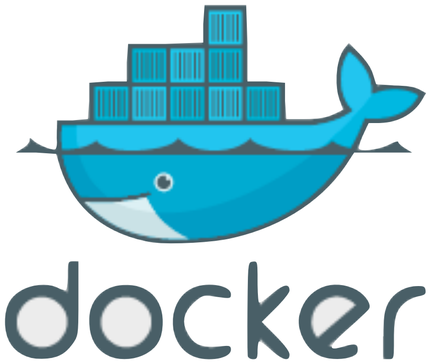

```
docker run -p 8081:8081 server:1.0.0

                Link the two containers

docker run --link server:server -p 80:80 frontend:1.0.0
```

Docker Compose

Previously Fig.sh

- We can specify the configuration in a config file
- A significant improvement over manual invocation

```
server:
  image: server:1.0.0
  ports:
    - "8081:8081"
frontend:
  image: frontend:1.0.0
  ports:
    - "80:80"
  links:
    - server:server
```



https://docs.docker.com/compose/

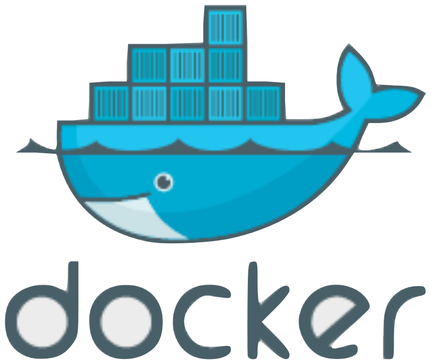
Docker Compose

Previously Fig.sh

- We can specify the configuration in a config file
- A significant improvement over manual invocation

```
server:
  image: server:1.0.0
  ports:
    - "8081:8081"
frontend:
  image: frontend:1.0.0
  ports:
    - "80:80"
  links:
    - server:server
```

Individual Container Configuration



https://docs.docker.com/compose/

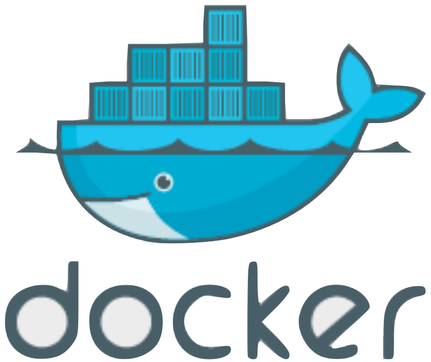
Docker Compose

Previously Fig.sh

- We can specify the configuration in a config file
- A significant improvement over manual invocation

```
server:
  image: server:1.0.0
  ports:
    - "8081:8081"
frontend:
  image: frontend:1.0.0
  ports:
    - "80:80"
  links:
    - server:server
```

Image Name and Version



https://docs.docker.com/compose/

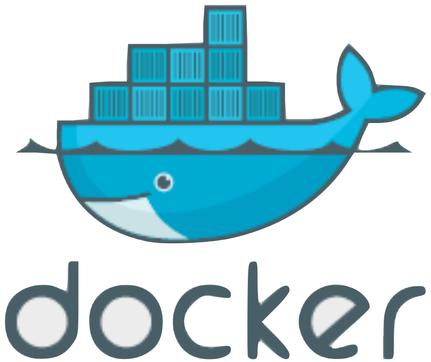
Docker Compose

Previously Fig.sh

- We can specify the configuration in a config file
- A significant improvement over manual invocation

```
server:
  image: server:1.0.0
  ports:
    - "8081:8081"
frontend:
  image: frontend:1.0.0
  ports:
    - "80:80"
  links:
    - server:server
```

Specify the Ports



https://docs.docker.com/compose/

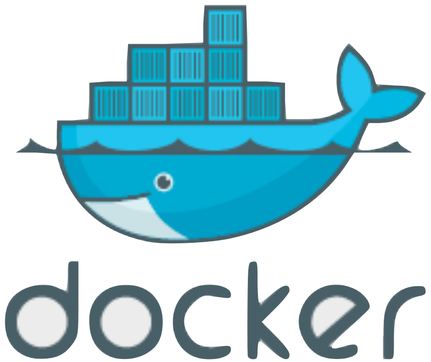
Docker Compose

Previously Fig.sh

- We can specify the configuration in a config file
- A significant improvement over manual invocation

```
server:
  image: server:1.0.0
  ports:
    - "8081:8081"
frontend:
  image: frontend:1.0.0
  ports:
    - "80:80"
  links:
    - server:server
```

Link the two Containers



https://docs.docker.com/compose/

docker-java

- Works on top of the Docker API
- Controls the containers from code
- Supports a range of configuration options

https://github.com/docker-java/docker-java

docker-java

```
DockerClientConfig config = DockerClientConfig.createDefaultConfigBuilder()
                                .withUri("unix:///var/run/docker.sock").build();
DockerClient client = DockerClientBuilder.getInstance(config).build();
String imageName = "frontend";
List<String> envStrs = new ArrayList<String>();

int linkCount = 1;
Link[] containerLinks = new Link[linkCount];
containerLinks[0] = new Link("server", "server");

String id = client
     .createContainerCmd(imageName)
     .exec()
     .getId();
final StartContainerCmd startContainerCmd = client
     .startContainerCmd(id)
     .withPublishAllPorts(true)
     .withLinks(containerLinks);

try {
   startContainerCmd.exec();
} catch (NotModifiedException e) {
   //swallow... this is fine!
}
```

https://github.com/docker-java/docker-java

docker-java

```
DockerClientConfig config = DockerClientConfig.createDefaultConfigBuilder()
                                .withUri("unix:///var/run/docker.sock").build();
DockerClient client = DockerClientBuilder.getInstance(config).build();
String imageName = "frontend";
List<String> envStrs = new ArrayList<String>();

int linkCount = 1;
Link[] containerLinks = new Link[linkCount];
containerLinks[0] = new Link("server", "server");

String id = client
     .createContainerCmd(imageName)
     .exec()
     .getId();
final StartContainerCmd startContainerCmd = client
     .startContainerCmd(id)
     .withPublishAllPorts(true)
     .withLinks(containerLinks);

try {
   startContainerCmd.exec();
} catch (NotModifiedException e) {
   //swallow... this is fine!
}
```

Individual Container Configuration

https://github.com/docker-java/docker-java

docker-java

```
DockerClientConfig config = DockerClientConfig.createDefaultConfigBuilder()
                                .withUri("unix:///var/run/docker.sock").build();
DockerClient client = DockerClientBuilder.getInstance(config).build();
String imageName = "frontend";
List<String> envStrs = new ArrayList<String>();

int linkCount = 1;
Link[] containerLinks = new Link[linkCount];
containerLinks[0] = new Link("server", "server");

String id = client
     .createContainerCmd(imageName)
     .exec()
     .getId();
final StartContainerCmd startContainerCmd = client
     .startContainerCmd(id)
     .withPublishAllPorts(true)
     .withLinks(containerLinks);

try {
   startContainerCmd.exec();
} catch (NotModifiedException e) {
   //swallow... this is fine!
}
```

Image Name

https://github.com/docker-java/docker-java

docker-java

```
DockerClientConfig config = DockerClientConfig.createDefaultConfigBuilder()
                                .withUri("unix:///var/run/docker.sock").build();
DockerClient client = DockerClientBuilder.getInstance(config).build();
String imageName = "frontend";
List<String> envStrs = new ArrayList<String>();

int linkCount = 1;
Link[] containerLinks = new Link[linkCount];
containerLinks[0] = new Link("server", "server");

String id = client
     .createContainerCmd(imageName)
     .exec()
     .getId();
final StartContainerCmd startContainerCmd = client
     .startContainerCmd(id)
     .withPublishAllPorts(true)
     .withLinks(containerLinks);

try {
   startContainerCmd.exec();
} catch (NotModifiedException e) {
   //swallow... this is fine!
}
```

Specify the Ports

https://github.com/docker-java/docker-java

docker-java

```
DockerClientConfig config = DockerClientConfig.createDefaultConfigBuilder()
                                .withUri("unix:///var/run/docker.sock").build();
DockerClient client = DockerClientBuilder.getInstance(config).build();
String imageName = "frontend";
List<String> envStrs = new ArrayList<String>();

int linkCount = 1;
Link[] containerLinks = new Link[linkCount];
containerLinks[0] = new Link("server", "server");

String id = client
     .createContainerCmd(imageName)
     .exec()
     .getId();
final StartContainerCmd startContainerCmd = client
     .startContainerCmd(id)
     .withPublishAllPorts(true)
     .withLinks(containerLinks);

try {
   startContainerCmd.exec();
} catch (NotModifiedException e) {
   //swallow... this is fine!
}
```

Link the Containers

https://github.com/docker-java/docker-java

Docker Compose

- Good middle ground
- Can be run from Jenkins, or code with exec
- Perfect for setting up environments
- Cost is low
- Benefit is high

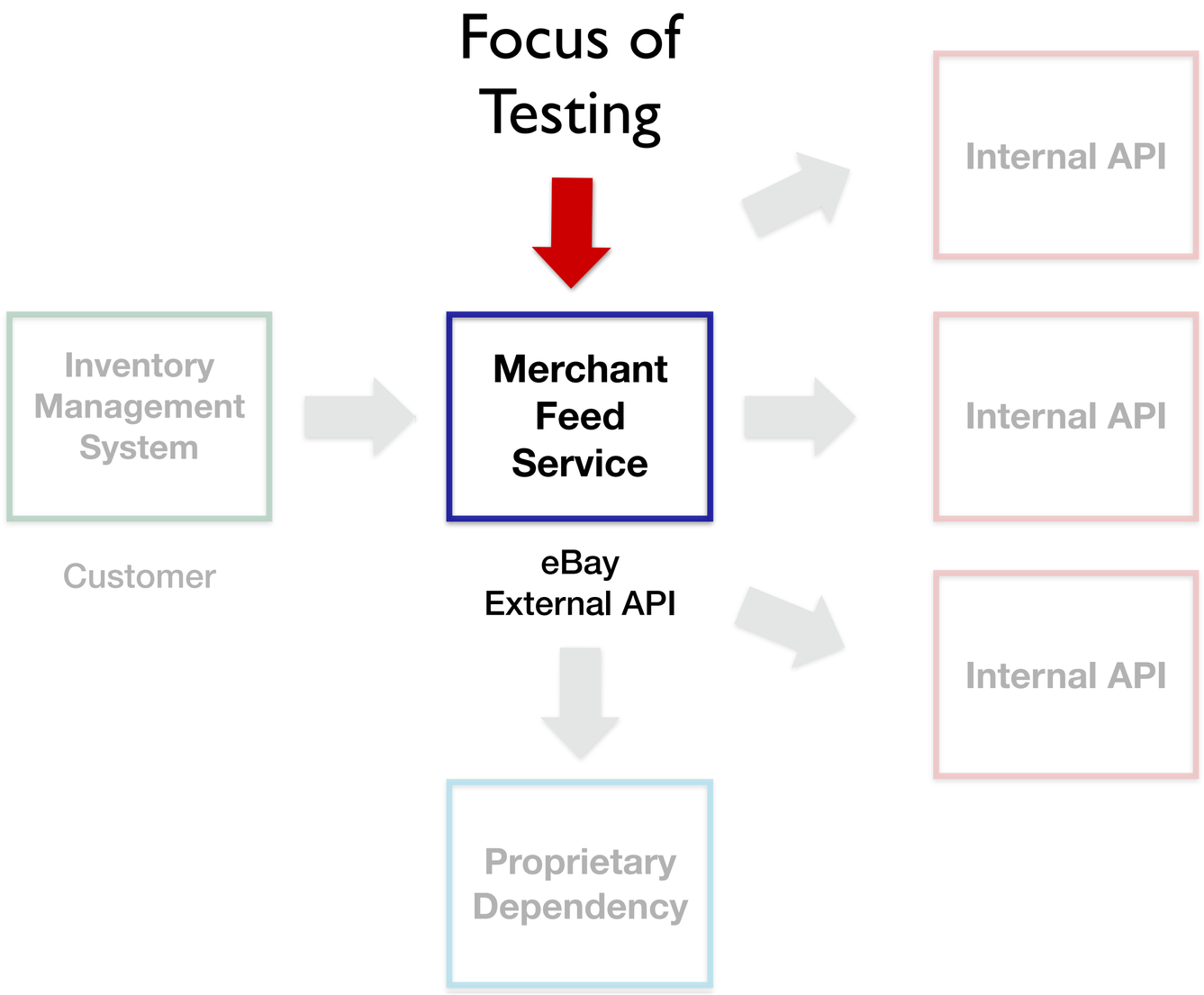
Image: https://www.flickr.com/photos/sergesegal/

The Desired Outcome

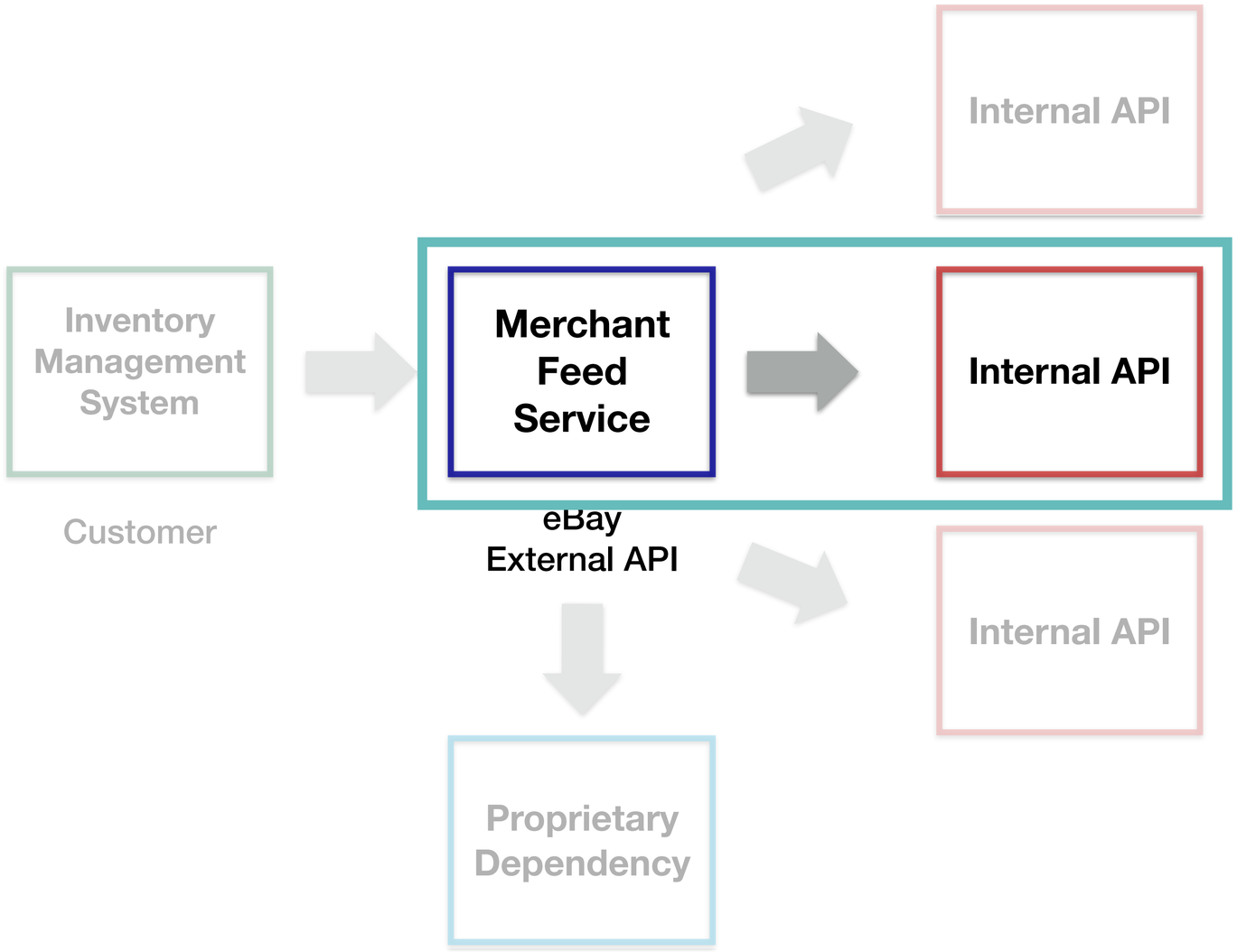
- Reduce the time it takes to run our end to end tests against our system

Image: https://www.flickr.com/photos/75680924@N08/

Isolation is good

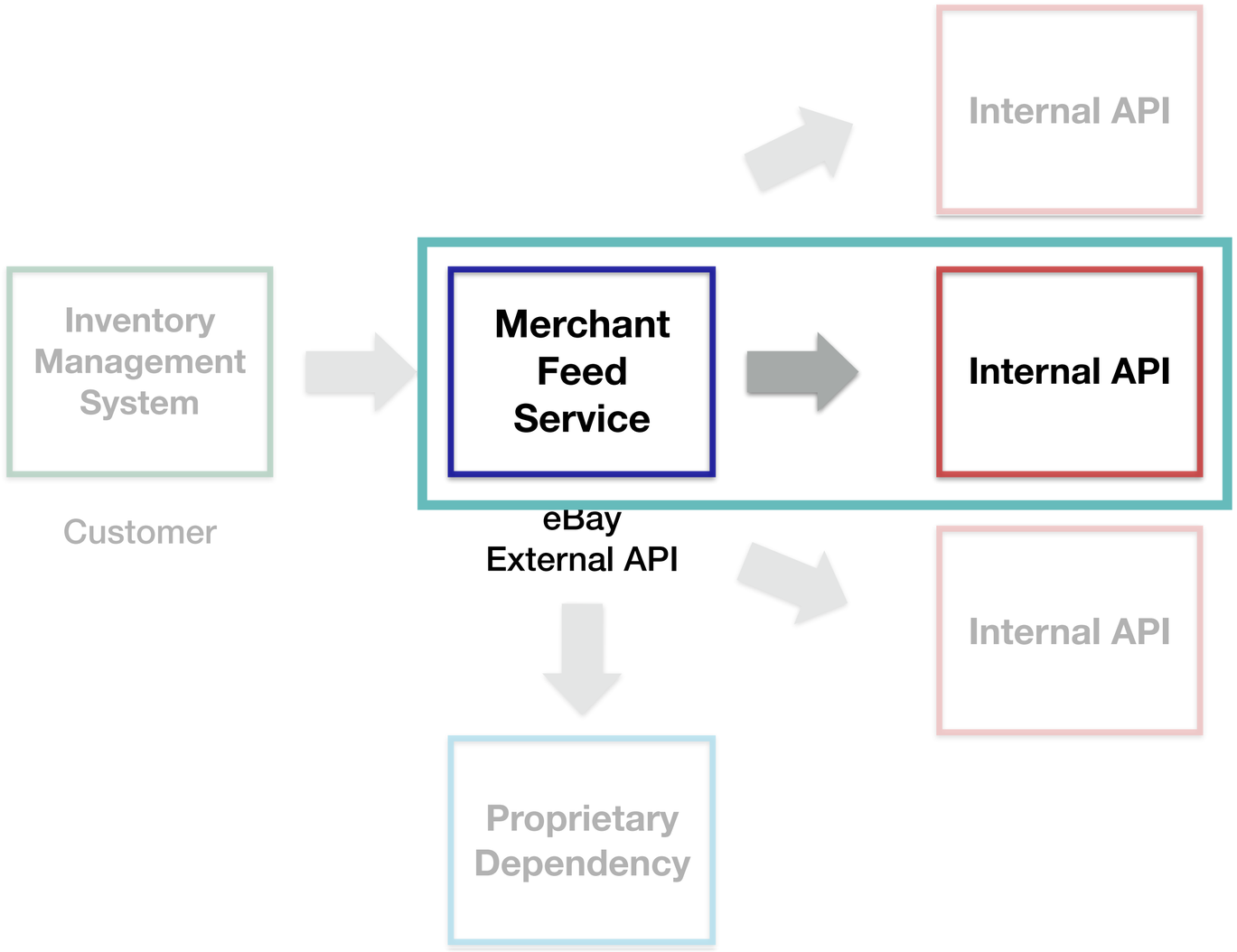


Isolation is good



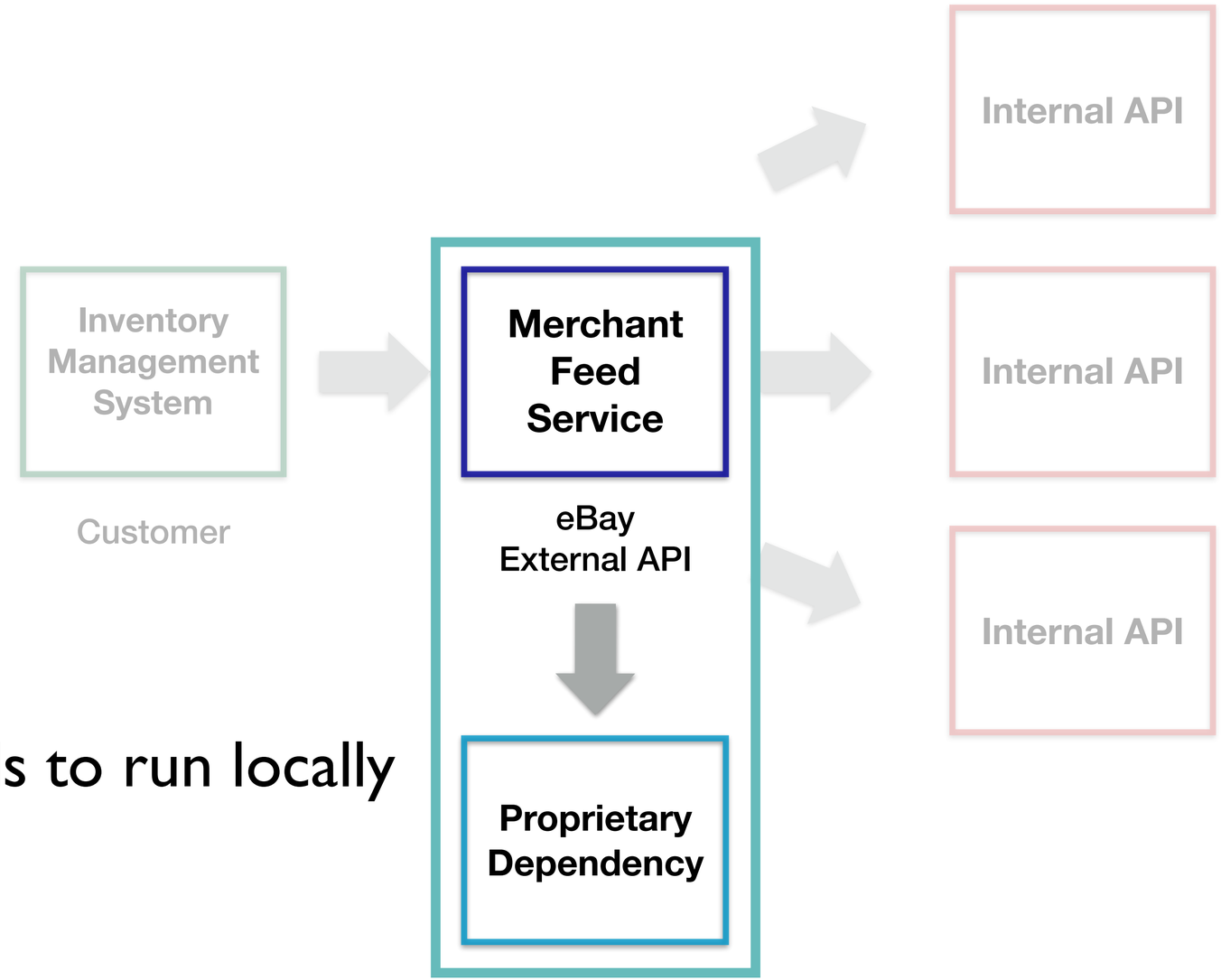
- Across the Atlantic

Isolation is good



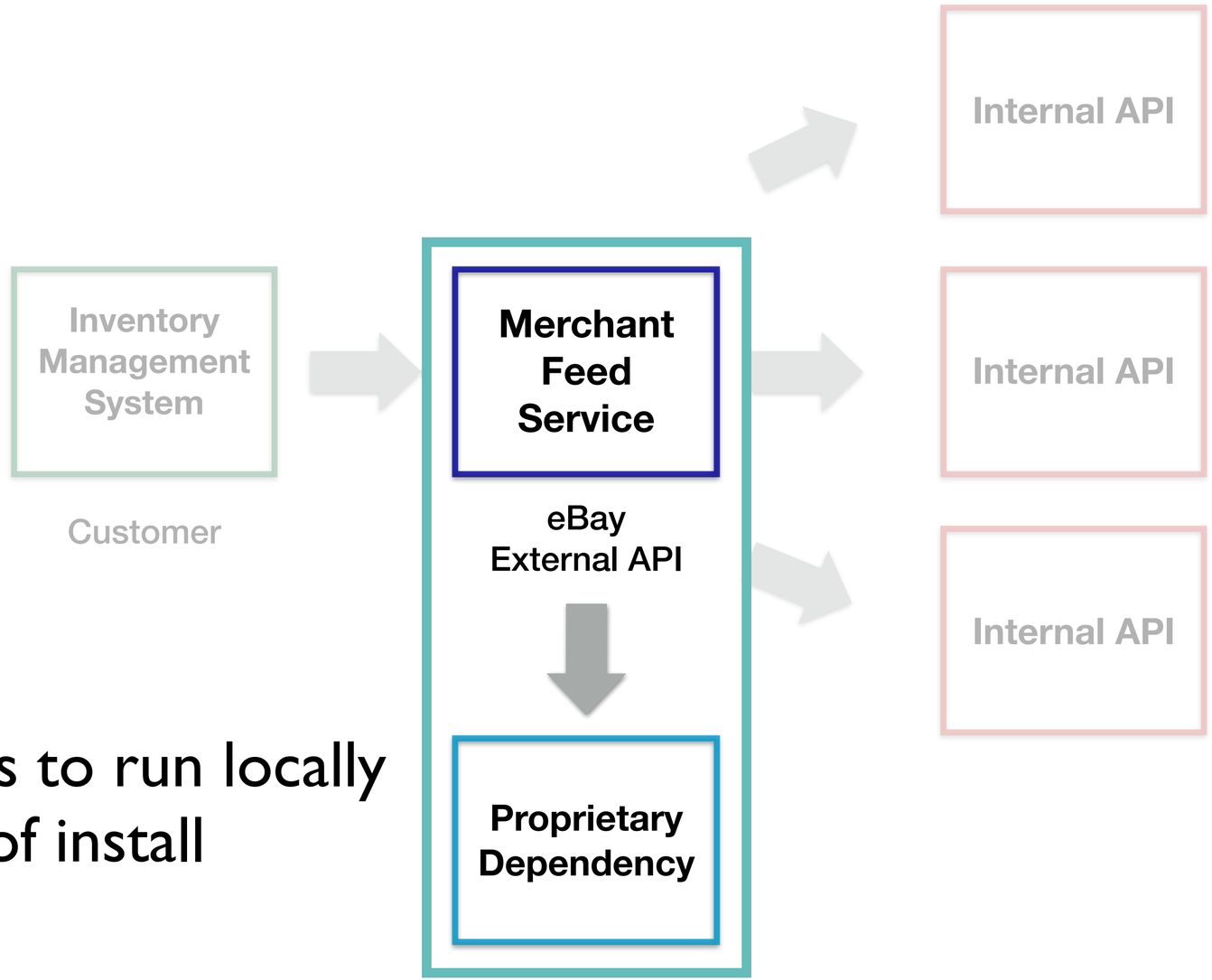
- Across the Atlantic
- High Latency

Isolation is good



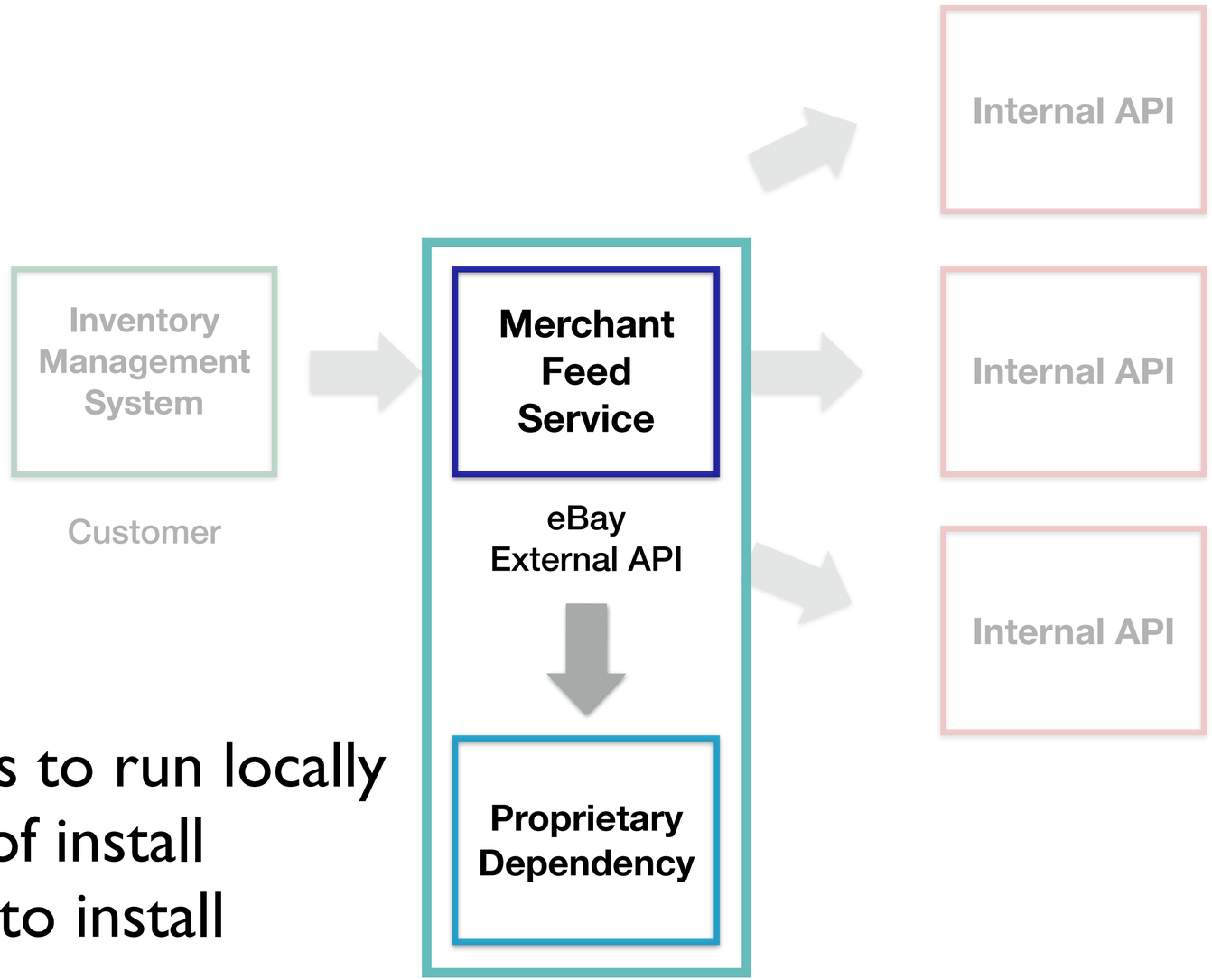
- Needs to run locally

Isolation is good



- Needs to run locally
- GB's of install

Isolation is good



- Needs to run locally
- GB's of install
- Hard to install

Capacity Testing

- A suite of tests for each resource

Image: https://www.flickr.com/photos/75680924@N08/

Capacity Testing

- A suite of tests for each resource
- Proprietary dependencies are put in containers, not VMs

Image: https://www.flickr.com/photos/75680924@N08/

Capacity Testing

- A suite of tests for each resource
- Proprietary dependencies are put in containers, not VMs
- Internal Dependencies are stubs in containers

Image: https://www.flickr.com/photos/75680924@N08/

Capacity Testing

- A suite of tests for each resource
- Proprietary dependencies are put in containers, not VMs
- Internal Dependencies are stubs in containers
  - Explicit control of input and output

Image: https://www.flickr.com/photos/75680924@N08/

Capacity Testing

- A suite of tests for each resource
- Proprietary dependencies are put in containers, not VMs
- Internal Dependencies are stubs in containers
  - Explicit control of input and output
  - Fixed Contract

Image: https://www.flickr.com/photos/75680924@N08/

Capacity Testing

- A suite of tests for each resource
- Proprietary dependencies are put in containers, not VMs
- Internal Dependencies are stubs in containers
  - Explicit control of input and output
  - Fixed Contract
  - As fast as we need them to be

Image: https://www.flickr.com/photos/75680924@N08/

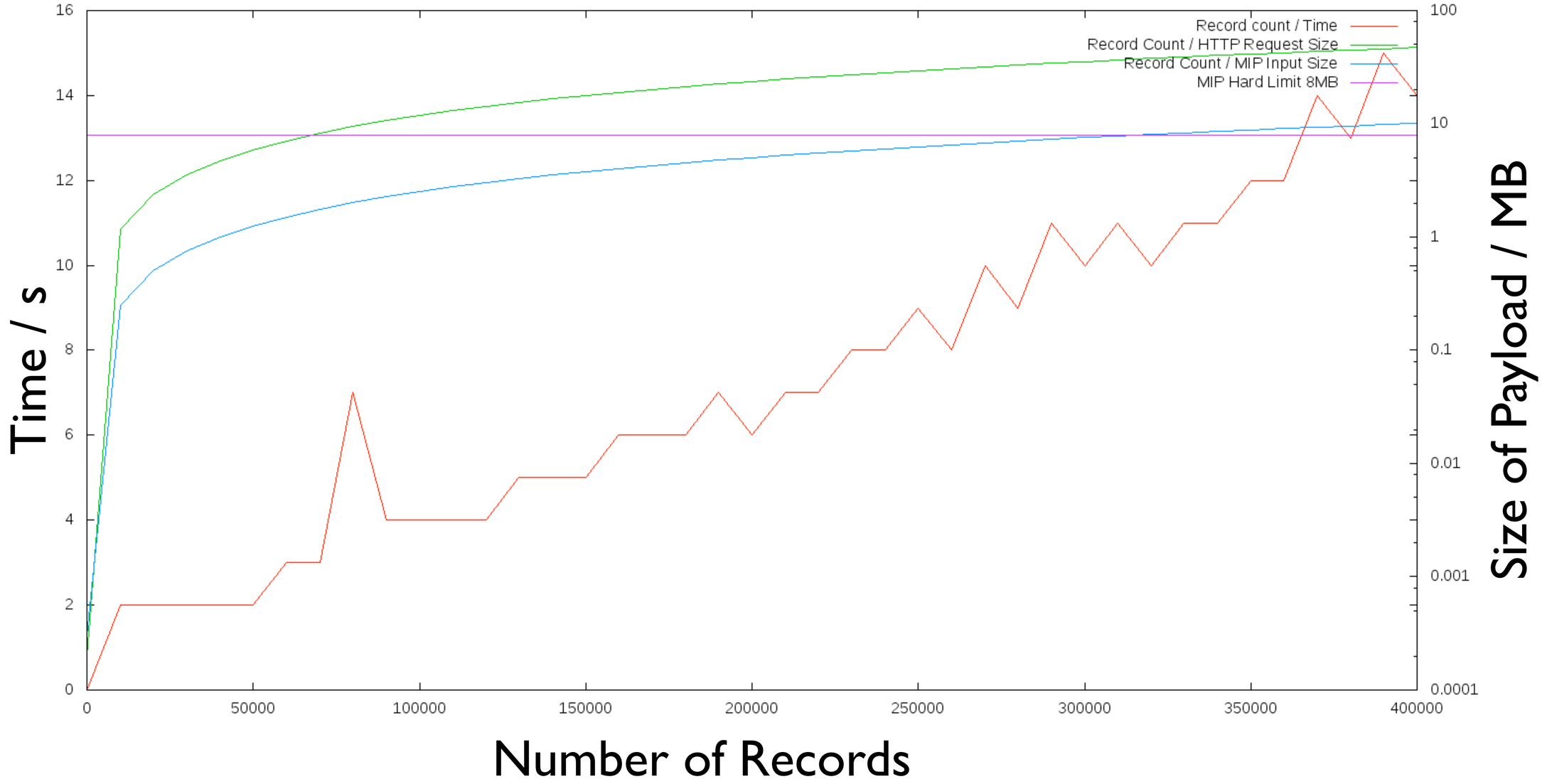
The Effect

- We cut our end to end testing time

120 minutes ⟶ 10 minutes

Image: https://www.flickr.com/photos/75680924@N08/

Isolation is good



Summary

- Stubs cut our end to end test time
- Docker allows us to easily replace dependencies for test scenarios
- Stubs fixed our contract with external dependencies

Image: https://www.flickr.com/photos/sergesegal/

THANK YOU!

Capacity Testing with Docker

Daniel Brown
@_dlpb

Questions?