

# Docker Compose

Um novo paradigma de desenvolvimento com docker compose

13 May 2015

Luciano Borguetti, Tiago Katcipis

**Infraestrutura de desenvolvimento**

# Necessidades

- Facilidade de construção de microserviços
- Rastreabilidade
- Facilitar execução de testes isolados e de integração
- Execução de testes o mais próximo possível do ambiente produção
- Independente de plataforma

**Docker**

# O que é?



- Virtualização em nível de sistema operacional ( $\geq$  release 0.9 **libcontainer** (<https://github.com/docker/libcontainer>), sem carga extra de um **hypervisor** (<http://en.wikipedia.org/wiki/Hypervisor>).
- Isolamento entre os containers utilizando Linux **namespaces** (<http://man7.org/linux/man-pages/man7/namespaces.7.html>)
- **Cgroups** (<https://www.kernel.org/doc/Documentation/cgroups/cgroups.txt>) para compartilhar e limitar recurso de hardware entre os containers
- **UnionFS** (<http://unionfs.filesystems.org/>) para criação de **layers** (<https://docs.docker.com/terms/layer/>), permitindo versionamento do sistema de arquivos dos containers

## Ciclo de vida de um container



# Docker Linking

# O que é?

- Permite que containers se conecte a outros containers.

```
docker run -d --name myredis redis
docker run --link myredis:myredis --volume `pwd`:/myapp --name myapp myapp
```

- As informações de conectividades são expostas de duas formas:

## Com a atualização do `/etc/hosts`

```
docker run ... cat /etc/hosts|grep -i myredis
172.17.0.69      myredis b181f946013f
```

## Através de variáveis de ambiente

```
docker run ... env|grep -i myredis
MYREDIS_PORT=tcp://172.17.0.69:6379
MYREDIS_PORT_6379_TCP=tcp://172.17.0.69:6379
MYREDIS_PORT_6379_TCP_ADDR=172.17.0.69
MYREDIS_PORT_6379_TCP_PORT=6379
MYREDIS_PORT_6379_TCP_PROTO=tcp
MYREDIS_NAME=/myapp/myredis
MYREDIS_ENV_REDIS_VERSION=3.0.0
```



# Exemplo

```
docker run -d --name myredis redis
docker run --link myredis:myredis --volume `pwd`: /myapp --name myapp myapp
```

## myapp.py

```
from flask import Flask
from redis import Redis
import os

app = Flask(__name__)
redis = Redis(host='myredis', port=6379)

@app.route('/')
def hello():
    redis.incr('hits')
    return 'Hello World! I have been seen %s times.' % redis.get('hits')

if __name__ == "__main__":
    app.run(host="0.0.0.0", debug=True)
```

## Docker vs Necessidades

- Rastreabilidade - **Ok**
- Execução de testes o mais próximo possível do ambiente produção - **Ok**
- Facilidade de execução de testes isolados - **Ok**
- Facilidade de execução de testes integração - **Não**
- Facilidade de construção de ambientes heterogêneos - **Não**

# Docker Compose

## O que é?

- Ferramenta para a definição e execução de serviços compostos por diversos containers.
- Possibilita expressar de forma declarativa a configuração de cada container e o relacionamento entre eles
- Expressa em apenas um lugar como diferentes containers se compoem para fornecer um serviço.

# Shell Script vs Docker Compose (YAML)

Iniciando uma serviço com uma aplicação web, e linkando com redis.

- Shell Script

```
#!/bin/bash
# Luciano Antonio Borguetti Faustino <lucianoborguetti@gmail.com>
# TDC 2015

# VARS
OPT=${1}

# Building myapp container image
function BuildContainers(){

    # Build image
    docker build -t myapp .

}

# Creating and linking myapp and myredis containers
function StartContainers(){

    # Run redis in backgroud
    docker run -d --name myredis redis

    # Run myapp container and link with myredis
    docker run --rm --name myapp --link myredis:myredis -v `pwd`:./myapp -p 5000:5000 my

}

}

# Removing myapp and myredis containers
function RemoveContainers(){
```

```
# Remove myapp container  
docker rm myapp
```

```
# Remove myredis container  
docker rm myredis
```

```
}
```

```
# Remove myapp container image  
function RemoveImages(){
```

```
# Remove myapp image  
docker rmi myapp
```

```
}
```

```
function ShowHelp(){
```

```
    echo "${0} [COMMAND]
```

```
    Commands:
```

```
    build      Build or rebuild services  
    rebuild   Rebuild services  
    help       Get help on a command  
    rm         Remove stopped containers  
    run        Run a one-off command  
    start      Start services  
    stop       Stop services  
    restart    Restart services  
    up         Create and start containers"
```

```
}
```

```
# Reading OPT and ...
```

```
case ${OPT} in
```

```
up)
```

```
    BuildContainers
```

```
    StartContainers
```

```
    ;;
```

```
start)
```

```
    StartContainers
```

```
    ;;
```

```
stop)
```

```
    StopContainers
```

```
    ;;
```

```
restart)
```

```
    StopContainers
```

```
    StartContainers
```

```
    ;;
```

```
rm)
```

```
    RemoveContainers
```

```
    ;;
```

```
build)
```

```
    BuildContainers
```

```
    ;;
```

```
rebuild)
```

```
    StopContainers
```

```
    RemoveContainers
```

```
    RemoveImages
```

```
    BuildContainers
```

```
    ;;
```

```
esac
```



```
ne1p)
```

```
    ShowHelp
```

```
    ;;
```

```
*)
```

```
    ShowHelp
```

```
    ;;
```

```
esac
```

# Shell Script vs Docker Compose (YAML)

Iniciando uma serviço com uma aplicação web, e linkando com redis.

- Docker Compose (YAML)

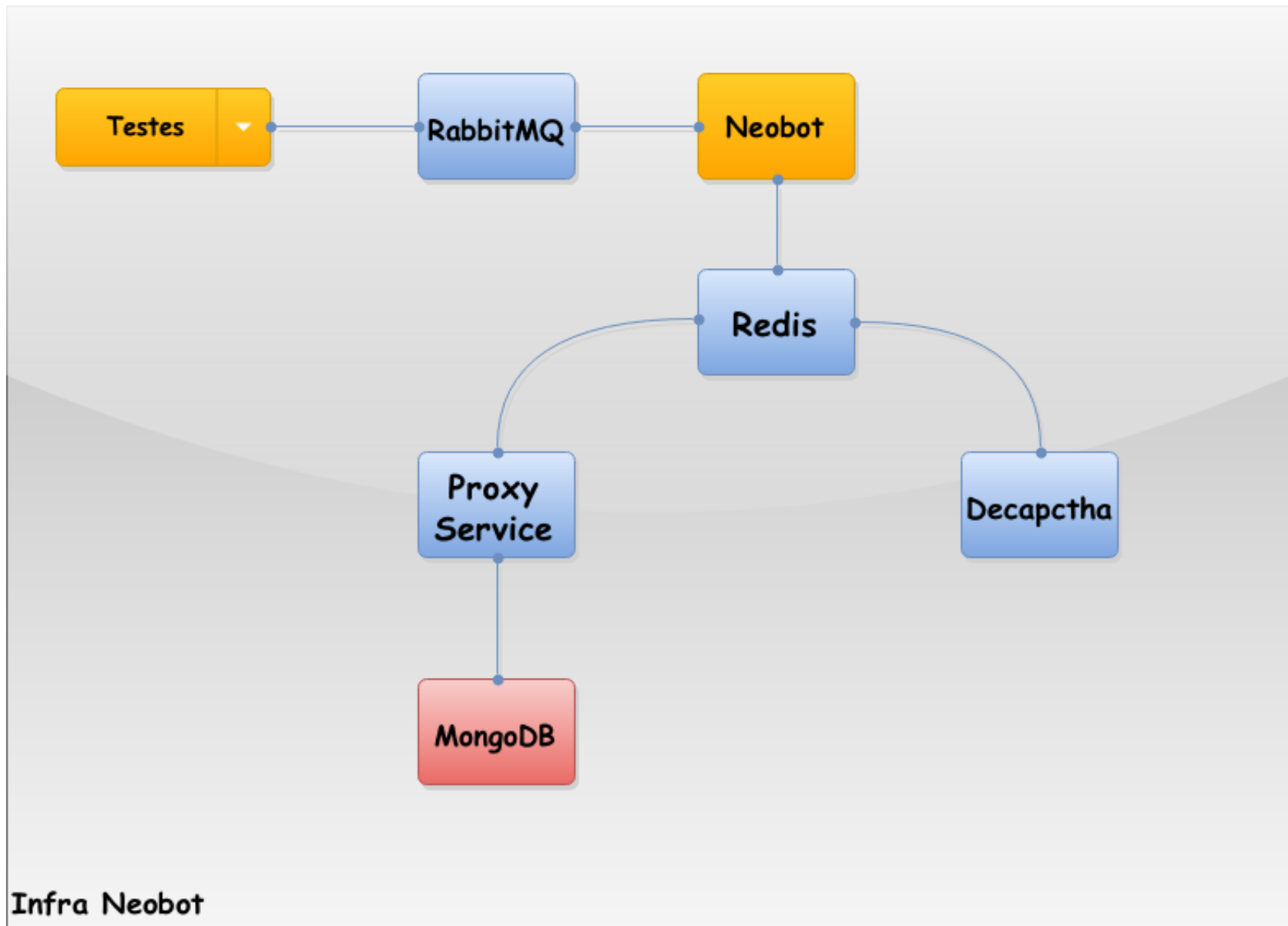
```
myapp:
  build: .
  links:
    - myredis
  volumes:
    - .:/myapp
  ports:
    - "5000:5000"
myredis:
  image: redis
```

**Caso de uso**

# Neobot

- Serviço de web scraping
- Basicamente uma extensão do [Scrapy](http://scrapy.org/) (<http://scrapy.org/>)
- Recebe jobs por meio de uma queue
- Extrai informações da fonte e disponibiliza essas informações em uma queue de resultados
- Resultados serão tratados por outro serviço
- Parece simples :-)

# Overview



# Dependências

- Precisamos de proxies válidos: proxyservice
- Proxies válidos se encontram em um MongoDB, precisamos integrar com ele
- Precisamos quebrar captchas: decaptcha
- Comunicação com esses serviços é feita com Redis
- API do Neobot é feita por filas, a implementação das filas usa RabbitMQ
- Um simples testes de integração envolve ter todos esses serviços executando e se comunicando

## Neobot: Objetivo

- Executar testes de integração, porém em um ambiente isolado
- O que seria isolamento ?
- Execução de um teste não deve influenciar a execução de outro
- Um desenvolvedor executando um teste não deveria interferir no trabalho de outro
- O ideal é executar todos os serviços necessários na máquina do desenvolvedor

**Resolvendo problema**



# Docker Compose File

```
neobot:
  command: ./hack/run.sh
  image: neobot-dev
  volumes:
    - ./neobot/dev
  ports:
    - "6080:6080"
    - "6023:6023"
  links:
    - proxyserver
    - redis
    - rabbitmq
    - decaptcher

decaptcher:
  image: registry.neoway.com.br/decaptcher
  links:
    - redis

proxyserver:
  image: registry.neoway.com.br/proxyserver
  links:
    - redis
  environment:
    MONGO_PORT: "tcp://<ip-address>:<port>"
    MONGO_USER: "XXX"
    MONGO_PASS: "YYY"

redis:
  .
  ..
```

image: redis

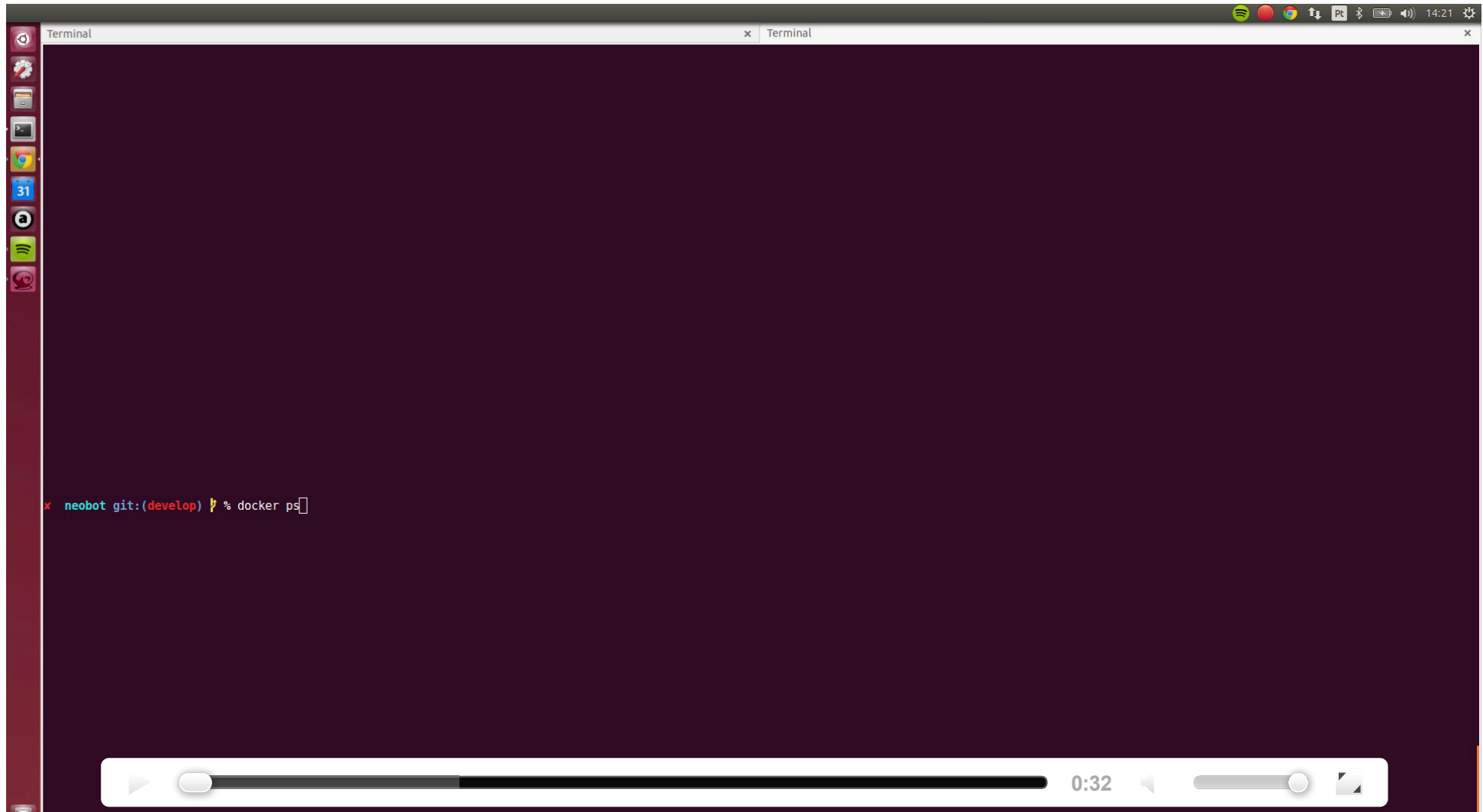
rabbitmq:

image: rabbitmq:management

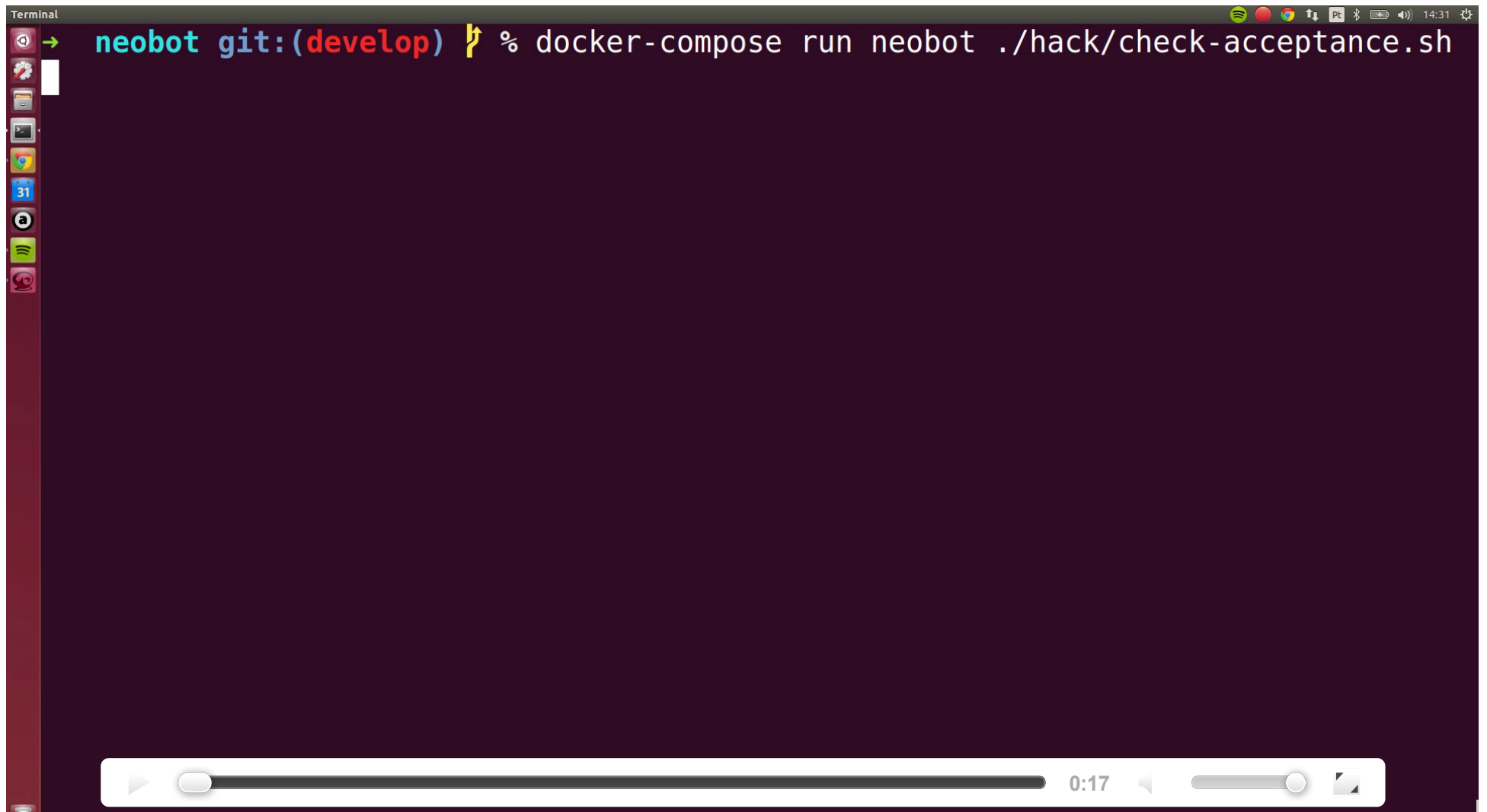
ports:

- "5672:5672"
- "8080:15672"

# Executando o daemon com todas as dependencias



# Executando testes integração



A terminal window titled "Terminal" is shown. The prompt is `neobot git:(develop) %`. The command entered is `docker-compose run neobot ./hack/check-acceptance.sh`. The terminal has a dark purple background. On the left side, there is a vertical dock with several application icons. At the bottom of the terminal window, there is a video player control bar with a play button, a progress slider, the time `0:17`, a volume icon, and a close button.

```
Terminal  
neobot git:(develop) % docker-compose run neobot ./hack/check-acceptance.sh
```

## Resultado

- Testes de integração envolvendo vários serviços
- Usando as mesmas imagens que serão usadas em produção
- Alto nível de isolamento (tudo executado localmente)
- Quantidade de ferramentas necessárias no host: 2 (docker e docker-compose)

# Mudança paradigma

- Testes de integração podem ser executados com um nível grande de isolamento
- Isso já era possível antes, mas não de um jeito automatizado e simples
- Nível de isolamento possibilita mais liberdade nos testes automatizados
- Por exemplo, filas do rabbit sempre são zeradas em cada teste
- Construção de ambientes de staging de maneira muito simples
- Dependências: docker + docker compose

# Thank you

Luciano Borguetti, Tiago Katcipsis

[lucianoborguetti@gmail.com](mailto:lucianoborguetti@gmail.com) / [tiago.katcipsis@neoway.com.br](mailto:tiago.katcipsis@neoway.com.br)

(<mailto:lucianoborguetti@gmail.com%20/%20tiago.katcipsis@neoway.com.br>)

<http://github.com/lborguetti> / <http://github.com/katcipsis>

(<http://github.com/lborguetti%20/%20http://github.com/katcipsis>)



