
Um olhar mais atento no Kubernetes

Whoami

- ★ Gerente na PagSeguro atuando com SRE/Cloud
- ★ +17 anos de experiência em Projetos FOSS

(Free and Open Source Software)

- ★ DevOps / InfraÁgil / SRE desde 2012
- ★ Principais Empresas:

pagseguro UOL stone 4LINUX FREE SOFTWARE SOLUTIONS EDS an HP company

- ★ Principais Certificações

Linux Professional Institute LPIC-OT DEVOPS TOOLS ENGINEER

LINUX FOUNDATION CERTIFIED ENGINEER

redhat CERTIFIED ENGINEER

Linux Professional Institute LPIC-2 LINUX ENGINEER

Objetivo

Container Runtime Interface
CRI

DESPERTAR EM VOCÊ SEU

“EU” QUESTIONADOR!

Evolução dos Containers

Unix

1979

Unix Chroot

Unix V7 introduziu a chamada do sistema chroot.

2000

FreeBSD Jails

O FreeBSD Jails permite particionar um sistema de FreeBSD em vários sistemas menores e independentes - chamados de "jails"

2005

OpenVZ

Similar ao FreeBSD Jails, é uma tecnologia de virtualização de nível de sistema operacional baseada no kernel e no sistema operacional do Linux.

2006

Process Containers

Lançado pelo Google foi projetado para limitar, contabilizar e isolar o uso de recursos (CPU, memória, I/O de disco, rede) de uma coleção de processos. Ele foi renomeado para "Grupos de Controle (**cgroups**)" um ano depois e acabou se fundindo ao kernel Linux 2.6.24.

2008

LXC

"LinuX Containers" foi a primeira e mais completa implementação do gerenciador de contêineres do Linux. Ele foi implementado usando namespaces cgroups e Linux e funciona em um único kernel Linux sem a necessidade de patches.

2003 - 2004

Borg - Sistema Interno do Google para gerenciamento de Cluster em larga escala.

2013 - Surge o Docker / 2014 - Surge o K8s

2013

Omega - "filho de Borg", foi impulsionado pelo desejo de melhorar a engenharia de software do ecossistema Borg.

Pycon
Mar/2013
dotCloud anuncia docker Open Source

Docker Inc.
Out/2013
dotCloud anuncia docker v0.1.0 Open Source

Docker 0.9
Mar/2014
Docker substitui o LXC pelo libcontainer

$$$
Abr/2015
A Docker foi estimada em mais de US $ 1 bilhão

dotCloud

Empresa de hospedagem fundada em 2010

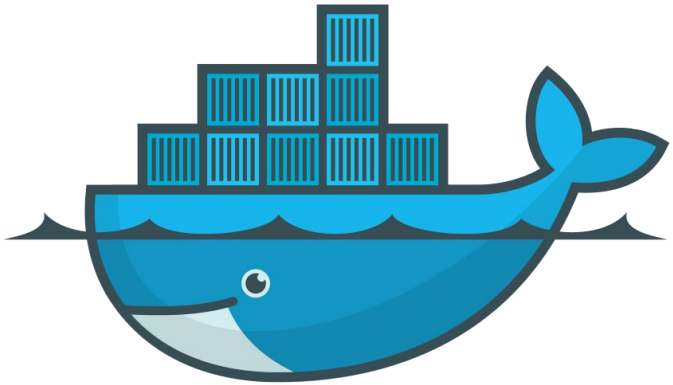
Parceria RedHat
Set/2013
Anunciam integração do Docker com a oferta PaaS do OpenShift Red Hat.

AWS ECS
Nov/2013
Amazon anuncia "EC2 Container Service"

Jun/2014

Kubernetes - Primeiro commit do Kubernetes

Casamento Perfeito!



Padronização (no Lock-in)

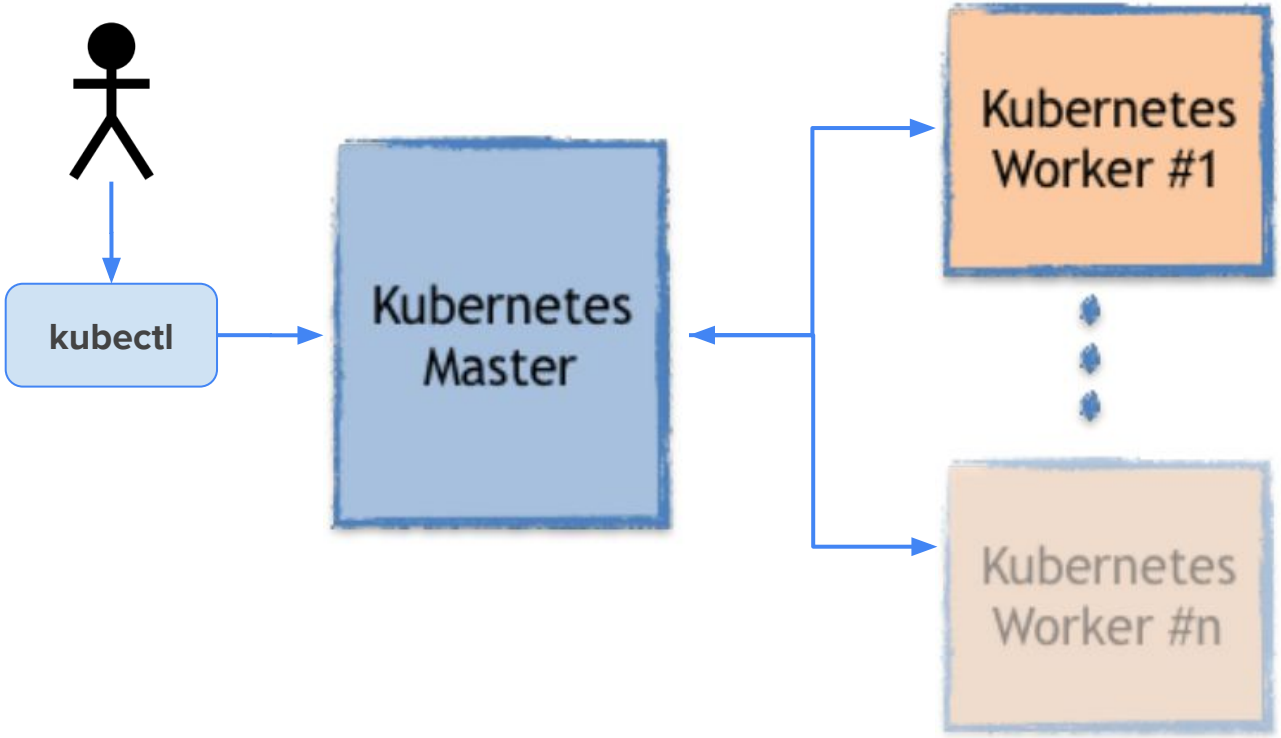
OCI
Jun/2015
Iniciativa da própria Docker

CNCF
Dez/2015
Parte da Linux Foundation.
Google “doa” kubernetes

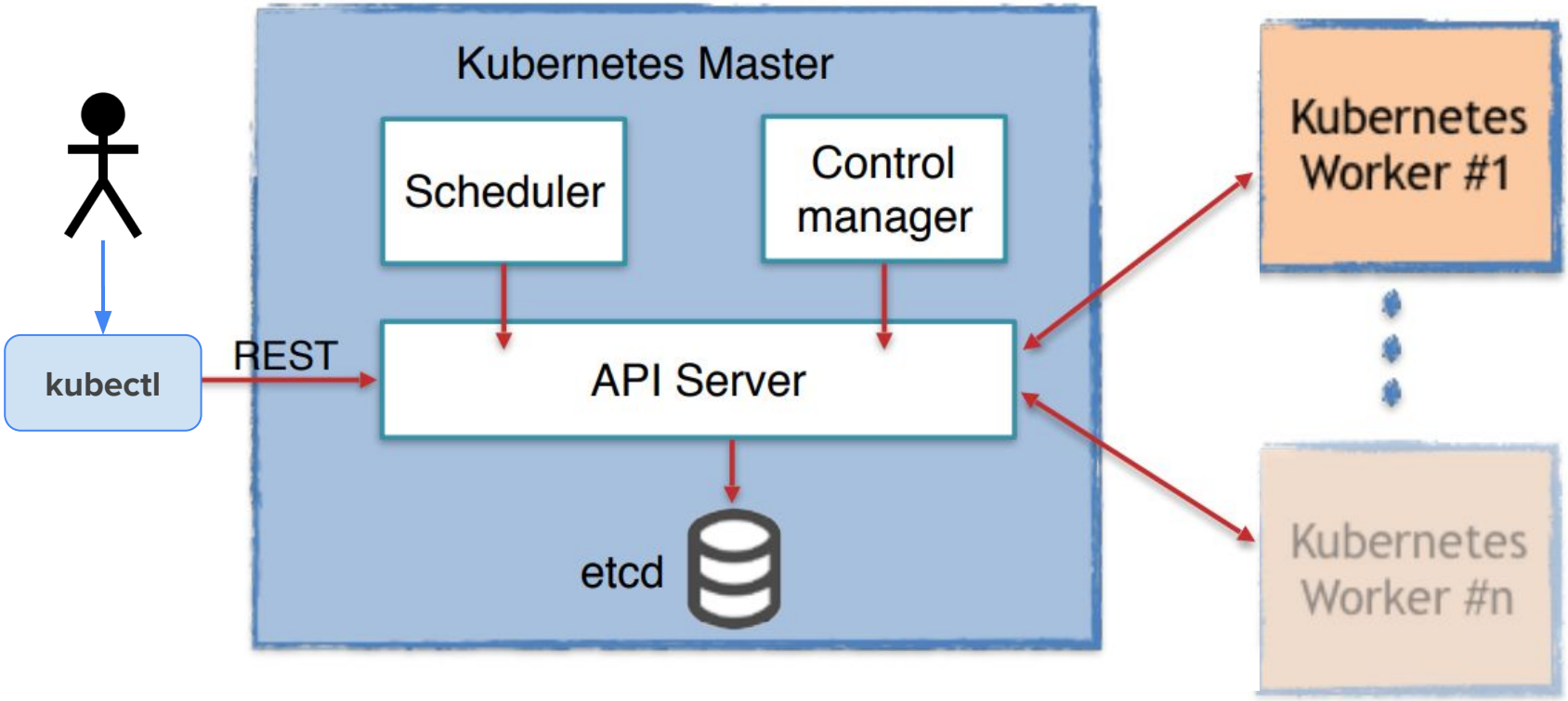
CLOUD NATIVE
COMPUTING FOUNDATION

Evolução do K8s

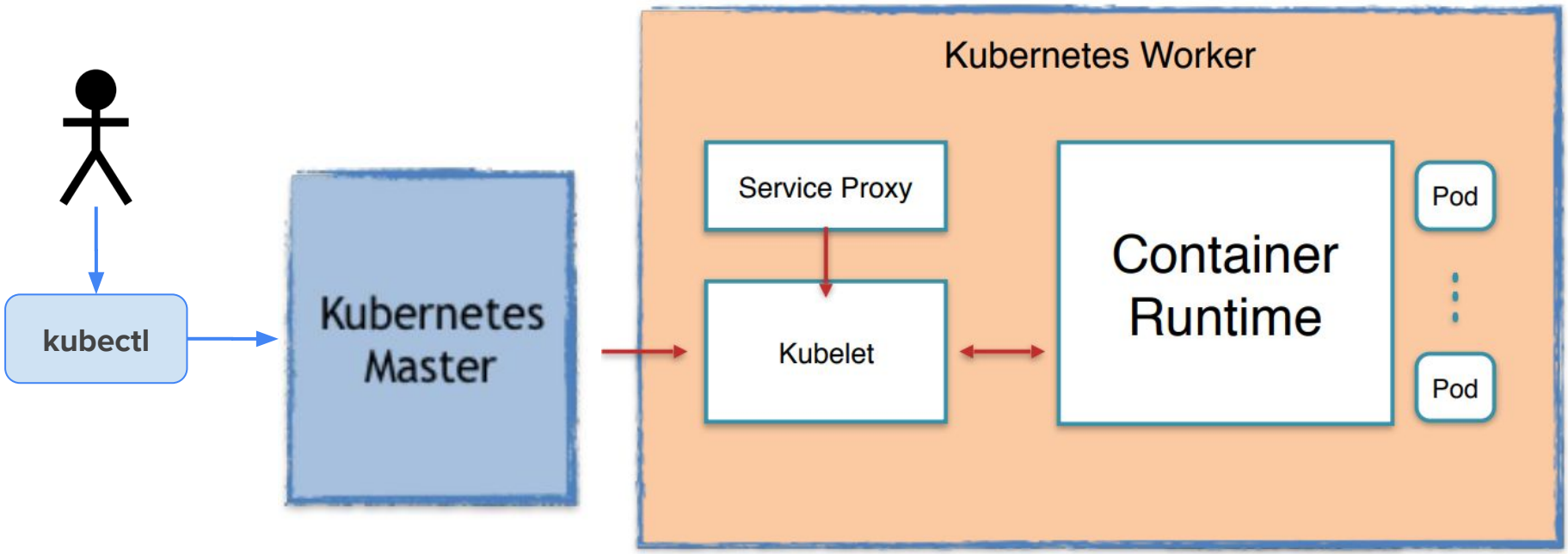
Arquitetura Básica



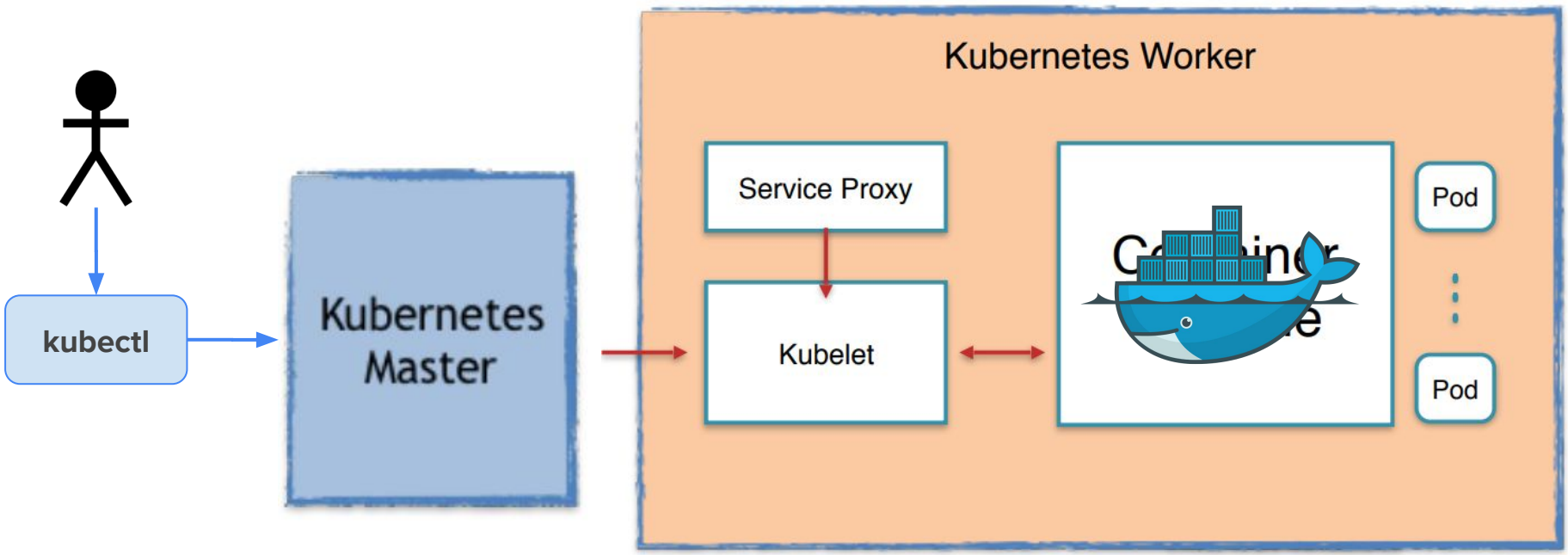
Arquitetura Básica



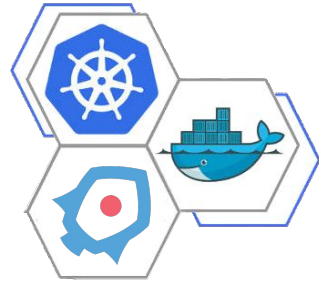
Arquitetura Básica

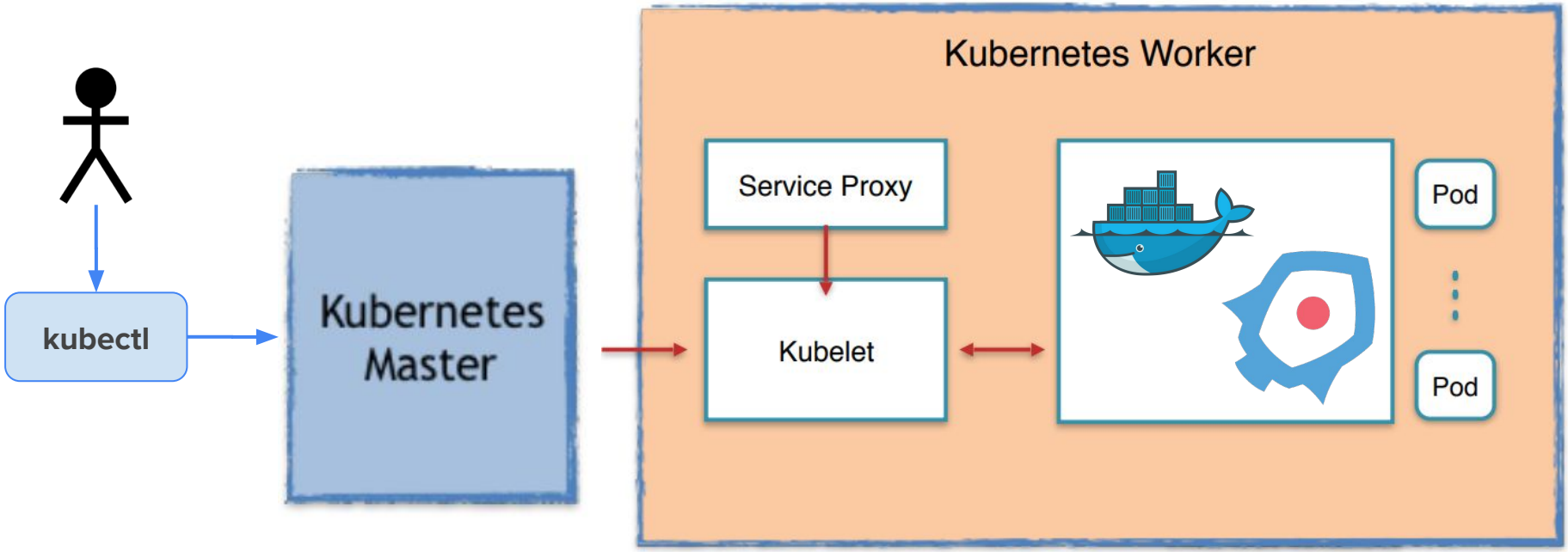


Arquitetura Básica

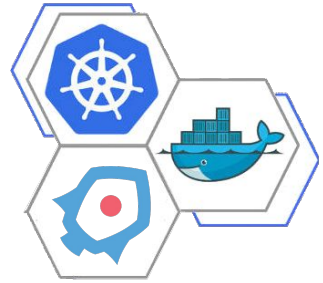


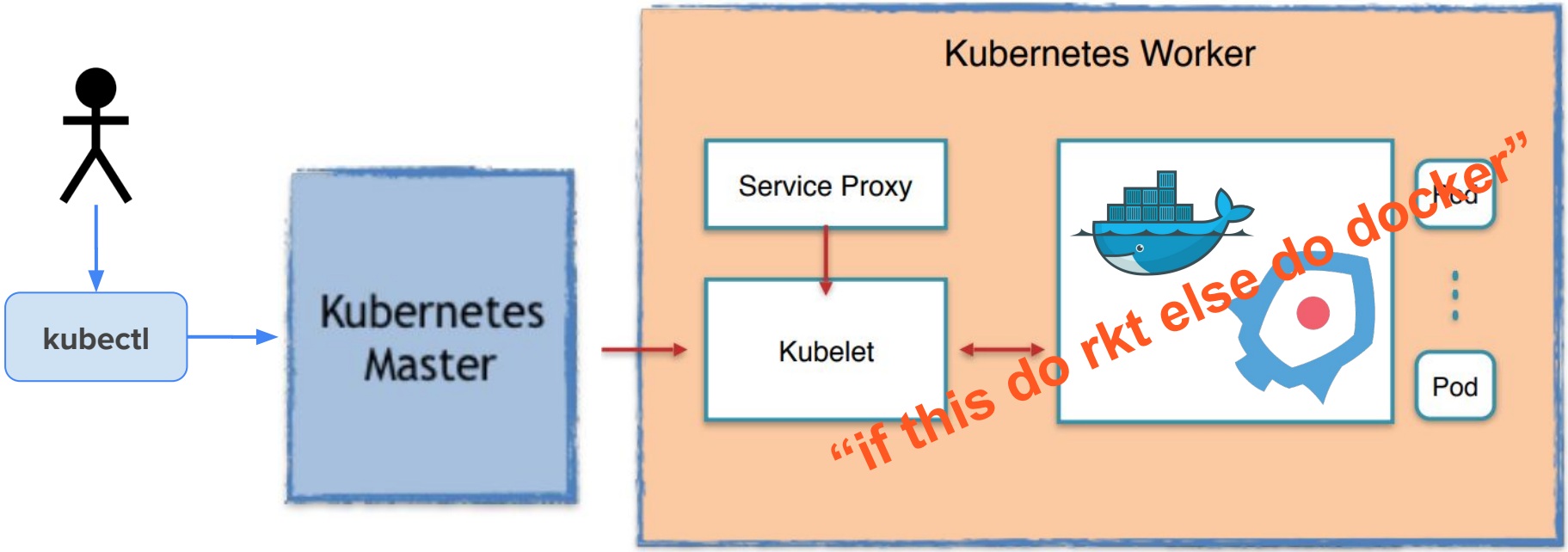
K8s 2014




kubectl
Kubernetes Master
Kubernetes Worker
Service Proxy
Kubelet
Pod
Pod

K8s 2014

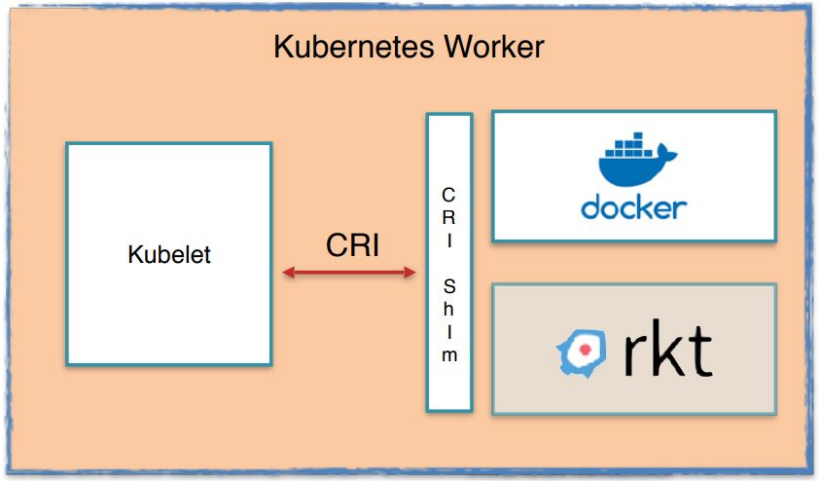



Kubernetes Worker
Service Proxy
Kubelet
kubectl
Kubernetes Master
Pod
Pod
“if this do rkt else do docker”

CRI

Containers Runtime Interface

Nem só de Docker vivem os containers...

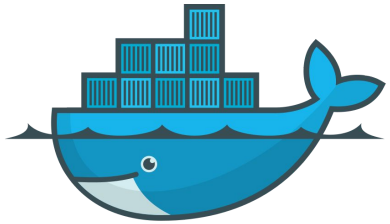


- O CRI é uma API que inclui dois serviços de gRPC, ImageService e RuntimeService.
- O ImageService fornece RPCs para extrair uma imagem de um repositório, inspecionar e remover uma imagem.
- O RuntimeService contém RPCs para gerenciar o ciclo de vida dos pods e contêineres, bem como chamadas para interagir com os contêineres (exec / attach / port-forward).

CRI

Containers Runtime Interface

Nem só de Docker vivem os containers...



kata containers

rkt

CoreOS

cri-o

redhat

Minikube

Containers Runtime Interface

https://minikube.sigs.k8s.io/docs/reference/runtimes/

Container Runtimes

Available container runtimes

Docker

The default container runtime in minikube is Docker. You can select it explicitly by using:

```
minikube start --container-runtime=docker
```

CRI-O

To use CRI-O:

```
minikube start --container-runtime=cri-o
```

containerd

To use containerd:

```
minikube start --container-runtime=containerd
```

Minikube / Container Runtime

```
gabycs@matrix:~/Documentos/TDC_Recife$ minikube start --container-runtime=docker -p docker --cpus=2 --memory=2048
😄  [docker] minikube v1.4.0 on Ubuntu 18.04
🔥  Creating virtualbox VM (CPUs=2, Memory=2048MB, Disk=20000MB) ...
🐳  Preparing Kubernetes v1.16.0 on Docker 18.09.9 ...
🚜  Pulling images ...
🚀  Launching Kubernetes ...
⌛  Waiting for: apiserver proxy etcd scheduler controller dns
🏄  Done! kubectl is now configured to use "docker"
```

```
gabycs@matrix:~/Documentos/TDC_Recife$ minikube start --container-runtime=cri-o -p crio --cpus=2 --memory=2048 --host-only-cidr='192.168.160.1/24'
😄  [crio] minikube v1.4.0 on Ubuntu 18.04
🔥  Creating virtualbox VM (CPUs=2, Memory=2048MB, Disk=20000MB) ...
🎁  Preparing Kubernetes v1.16.0 on CRI-O 1.15.2 ...
🚜  Pulling images ...
🚀  Launching Kubernetes ...
⌛  Waiting for: apiserver etcd scheduler controller
🏄  Done! kubectl is now configured to use "crio"
```

A era Kubernetes

- Nós, profissionais de TI, precisamos dominar uma única plataforma para orquestração de containers para ser relevante a 90% do mercado de trabalho.
- Usando Kubernetes estamos “livres” do famoso “Vendor Lock in”
- O kubernetes tornou-se a nova camada de portabilidade da aplicação, ele é o denominador comum entre tudo e todos.
- Essa “nova camada” nos leva a novos padrões de design para o desenvolvimento de aplicações modernas.

Portanto, a maneira como devemos olhar para o Kubernetes é mais como um paradigma fundamental que tem implicações em múltiplas dimensões, ao invés de apenas um orquestrador.

Mas....

CRI

Adicionando a padronização nos containers runtime conectáveis no Kubernetes por meio do CRI, podemos imaginar um mundo em que os **desenvolvedores e operadores possam escolher as ferramentas e stacks que funcionam para eles e experimentem a interoperabilidade em todo o ecossistema de containers.**

- Um desenvolvedor no MacBook usa **Docker para Mac** para desenvolver, e usa o **suporte do Kubernetes (usando Docker como CRI runtime)** para tentar implantar sua nova aplicação com Kubernetes pods.
- A **aplicação** vai através de algum **CI/CD** que usa **runc** para criar a imagem **OCI** e publicar no repositório corporativo de imagens para ela ser testada.
- Um cluster Kubernetes local usando **containerd como CRI runtime** roda o conjunto de testes contra a aplicação.
- Essa empresa escolhe utilizar **Kata containers** para alguns ambientes de produção, e quando a aplicação é implantada executa os pods com **containerd**, configurados para usar **Kata containers como runtime ao invés do runc**.

Todo este cenário exemplificado funciona perfeitamente devido à conformidade com a especificação OCI para runtime e imagens, e porque o CRI permite essa flexibilidade na escolha do runtime.

Thanks!

gabydias@gmail.com
https://www.linkedin.com/in/gabydias/