

# Habilitando Aplicações Nativas de Nuvem

## Introdução a Contêineres e Kubernetes



### 4. Kubernetes



# Programa: Introdução a Contêineres e Kubernetes



## 1. Conceitos Básicos

- ✓ Abstrações em Ciência da Computação
- ✓ Virtualização de Computadores
- ✓ MicroVMs e Unikernels



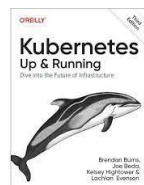
## 2. Contêineres

- Origem
- Fundamentos
- Criação e execução
- Registro e reuso
- Infraestrutura como Código
- Aplicativos Modernos



## 3. Kubernetes

- Origem
- Arquitetura
- Pods
- Abstrações de Recursos
  - **Descoberta de Serviços**
  - **Serviços de Rede**
  - **Instalação e administração básica**
  - **Implantação de um caso de uso (exemplo)**



Kubernetes Up & Running  
Brendan Burns, Joe Beda, Kelsey  
Hightower, and Lachlan Evenson  
Cortesia da VMware Inc.

# Refrescando nossa memória

- Nas aula passada estudamos os princípios fundamentais as abstrações de recursos mais importantes do K8s que comumente são usadas nos manifestos (arquivos YAML):
  - *apiVersion*
  - *kind*
  - *metadata*
  - *Spec*
- Estudamos também o uso de *Labels* (rótulos que usam pares de chave/valor) e *Annotations* (metadados adicionais)
- Atividades preparatórias para a aula de hoje: ler capítulos 6 e 7 do livro “Kubernetes Up and Running”
  - 6. Labels and Annotations
  - 7. Service Discovery

# Descoberta de Serviços

# Expondo um *Pod*

- A criação de um *pod*, com um servidor web *nginx* por exemplo, permite a especificação de uma porta TCP/IP.
- Esta porta permite o acesso ao *pod* a partir de qualquer nó do cluster.
  - Como por exemplo acessar qualquer nó através de SSH e usar *curl* com ambos endereços IPs.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: my-nginx
spec:
  selector:
    matchLabels:
      run: my-nginx
  replicas: 2
  template:
    metadata:
      labels:
        run: my-nginx
    spec:
      containers:
      - name: my-nginx
        image: nginx
        ports:
        - containerPort: 80
```

# Serviço 1/2

## *Analogia Gado versus Animais Domésticos*

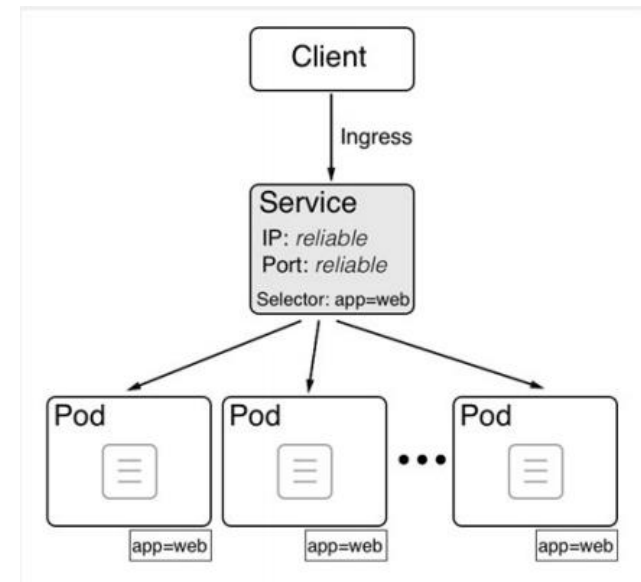
- No passado quando todos os sistemas eram de escala reduzida, cada servidor tinha um nome.
- E sabíamos exatamente que software estava sendo executado em cada servidor.
- Os *Pods* são executados em um espaço amplo e plano de endereços do cluster.
  - Em teoria todos podem ser acessados diretamente com <IP:port>.
  - Na prática é inviável administra-los assim.

## *Por que Serviços?*

- Quando um nó K8s fica indisponível: interrompe a execução de todos os *Pods* naquele nó.
- O cluster se encarregará de implantar novos *Pods* em outro(s) nós e terão portanto diferentes endereços IPs.

# Serviço 2/2

- Um *Serviço K8s* é uma abstração que define um conjunto lógico de *Pods* executando em qualquer nó do cluster, todos fornecendo a mesma funcionalidade.
- Um grupo de *Pods* (determinados por um mesmo *Label*) que tem o mesmo nome e número de portas e que servem a um mesmo propósito. Exemplo: Um *pool* de servidores HTTP com o mesmo conteúdo disponível.
- E um grupo de políticas de acesso.
- Este padrão as vezes é chamado de micro serviço.



# Exemplo de um manifesto de Serviço

```
apiVersion: v1
kind: Service
metadata:
  name: my-service
spec:
  selector:
    app: MyApp
  ports:
    - protocol: TCP
      port: 80
      targetPort: 9376
```

- Conjunto de *Pods* que usam a porta TCP 9376 com *Label* *app=MyApp*.
- Esse manifesto cria um novo *Serviço* denominado “*my-service*” disponível na porta TCP 9376 em qualquer *Pod* onde *app=MyApp*.





# Disponibilidade e equilíbrio de cargas

- Cada novo *Serviço* recebe um endereço IP único, mais conhecido como Cluster IP.
- Esse endereço é fixo, não muda durante a existência do Serviço.
- Todo e qualquer outro *Pod* pode usar este *Serviço*, sendo toda a comunicação encaminhada a um dos *Pods* membros do Serviço.

# Descoberta de Serviços

- De maneira nativa, a descoberta de serviços é feita através das K8s APIs:

```
$ kubectl create deployment alpaca-prod \  
  --image=gcr.io/kuar-demo/kuar-amd64:blue \  
  --port=8080  
$ kubectl scale deployment alpaca-prod --replicas 3  
$ kubectl expose deployment alpaca-prod  
$ kubectl create deployment bandicoot-prod \  
  --image=gcr.io/kuar-demo/kuar-amd64:green \  
  --port=8080  
$ kubectl scale deployment bandicoot-prod --replicas 2  
  kubectl expose deployment bandicoot-prod  
$ kubectl get services -o wide
```

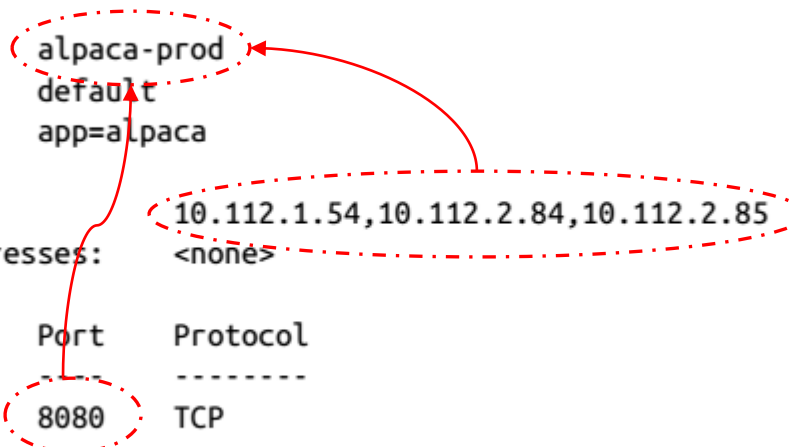
NAME	CLUSTER-IP	...	PORT(S)	...	SELECTOR
alpaca-prod	10.115.245.13	...	8080/TCP	...	app=alpaca
bandicoot-prod	10.115.242.3	...	8080/TCP	...	app=bandicoot
kubernetes	10.115.240.1	...	443/TCP	...	<none>

# Endpoints

- Alguns aplicativos e mesmo o sistema K8s necessitam ser capazes de acessar/usar serviços sem referenciar diretamente seus Cluster IPs.
- Isto é possível através de objetos *Endpoints* que são criados pelo K8s associando um nome (*endpoint*) aos endereços IPs e porta de acesso de um serviço:

```
$ kubectl describe endpoints alpaca-prod
```

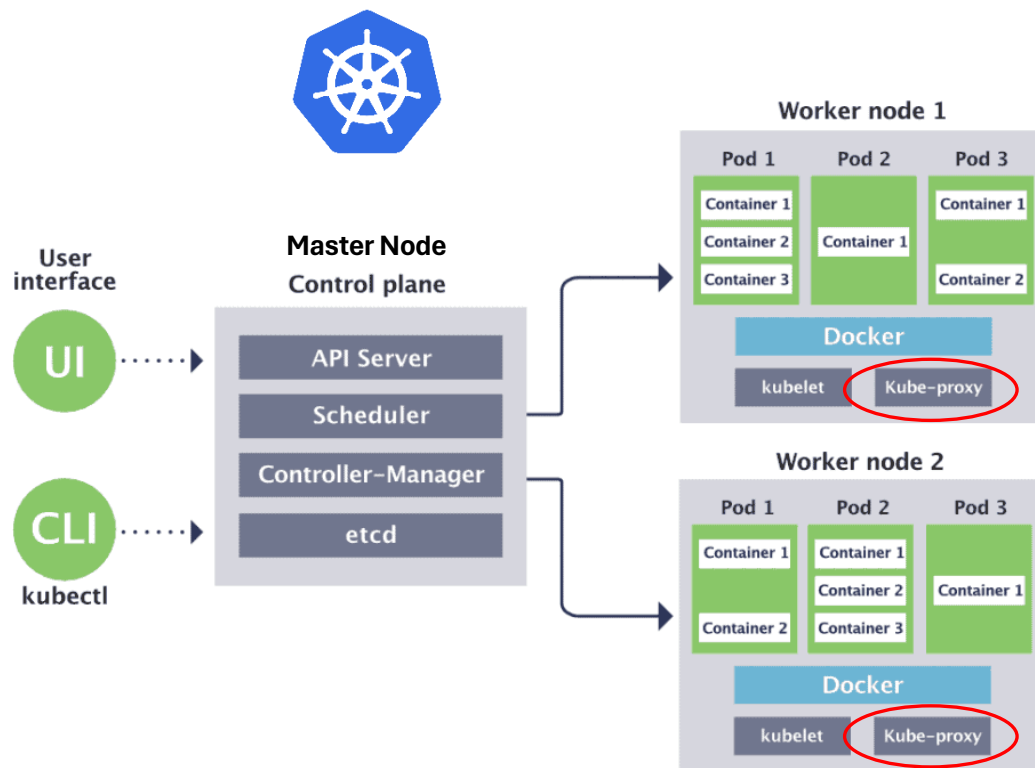
```
Name: alpaca-prod
Namespace: default
Labels: app=alpaca
Subsets:
  Addresses: 10.112.1.54,10.112.2.84,10.112.2.85
  NotReadyAddresses: <none>
  Ports:
    Name      Port      Protocol
    ----      -
    <unset>    8080      TCP
```



```
No events.
```

# Serviços de Rede

# IPs virtuais e proxies para serviços



- Em cada nó do cluster K8s existe uma instância de *kube-proxy* que se encarrega de monitorar a criação e a remoção objetos de *Serviços* e *End Points* pelo *Control Plane*.
- O *kube-proxy* é responsável por implementar uma forma de endereçamento virtual para os Serviços.

# Disponibilidade e equilíbrio de cargas

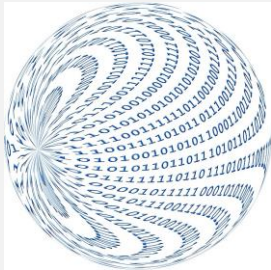
- Cada novo *Pod* obtém seu próprio endereço IP.
- Não é necessário criar links explícitos entre *Pods* e raramente é necessário lidar com o mapeamento entre portas de contêineres e portas de hosts.
- Os *Pods* podem comunicar-se com outros *Pods* independentemente do nó onde estão sendo executados.
- E por que não usar o serviço *DNS* em modo *round robin*?
  - Algumas implementações de *DNS* não respeitam *TTL*.
  - Os aplicativos demandam *TTL*.
  - *TTL* pequeno ou zero nos registros de *DNS* impõe uma alta carga no serviço *DNS*.



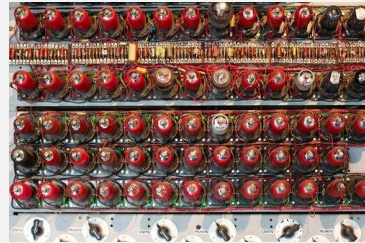


# Conclusão

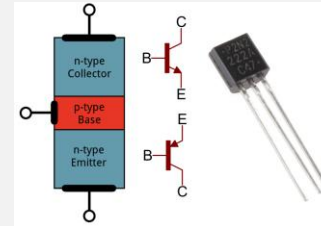
# A evolução das abstrações na Computação Moderna 1/3



Bits e Bytes



Computador  
Válvulas



Transistor



Computador  
Semicondutores

# A evolução das abstrações na Computação Moderna 2/3



Redes  
de  
Computadores



Centro de Dados



Virtualização



Centro de Dados  
Definido por Software  
(SDDC)

# A evolução das abstrações na Computação Moderna 3/3



# O que são Aplicativos Nativos de Nuvem?

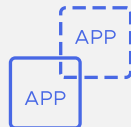
A CNCF<sup>2</sup> define a computação nativa de nuvem como o uso de software de código aberto, tecnologias como contêineres, micro serviços e malhas de serviços, para desenvolver e implantar aplicativos escalonáveis em plataformas de computação na nuvem.



Contêineres



Micro serviços



Escalonamento  
Horizontal



Malha  
de  
Serviços



Implantações  
Transparentes  
"Zero  
Downtime  
Deployment"



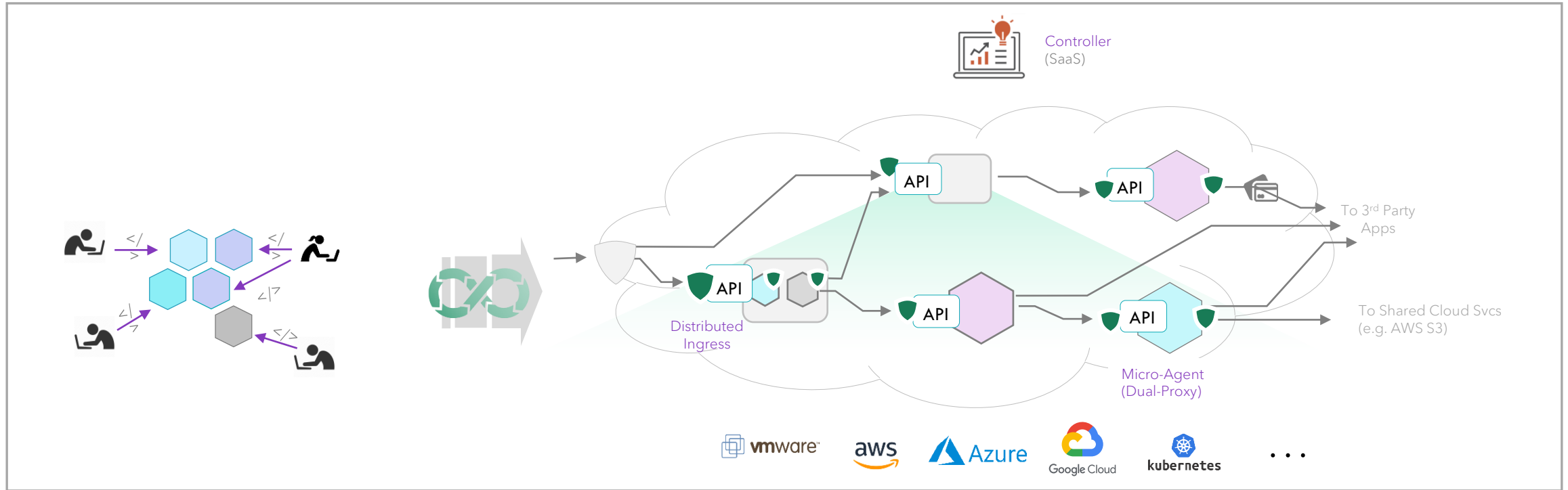
Integrações e  
Implantações  
Contínuas  
(CI/CD)



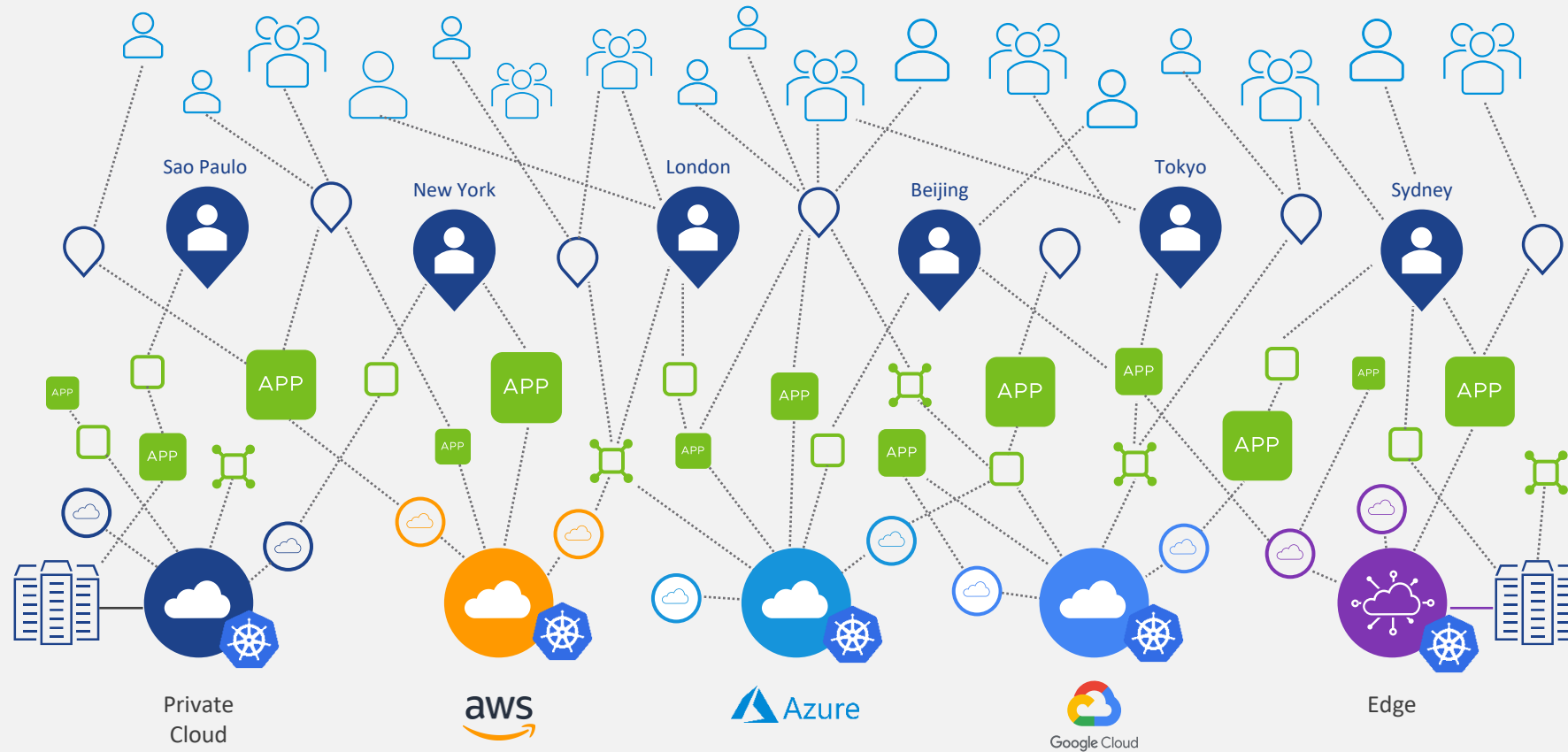
Orientação  
por APIs

1 - As part of the Linux Foundation, CNCF is the open source, vendor-neutral hub of cloud native computing, hosting projects like Kubernetes and Prometheus to make cloud native universal and sustainable, <https://www.cncf.io/>

# Agilidade, Resiliência e Inovação Continua

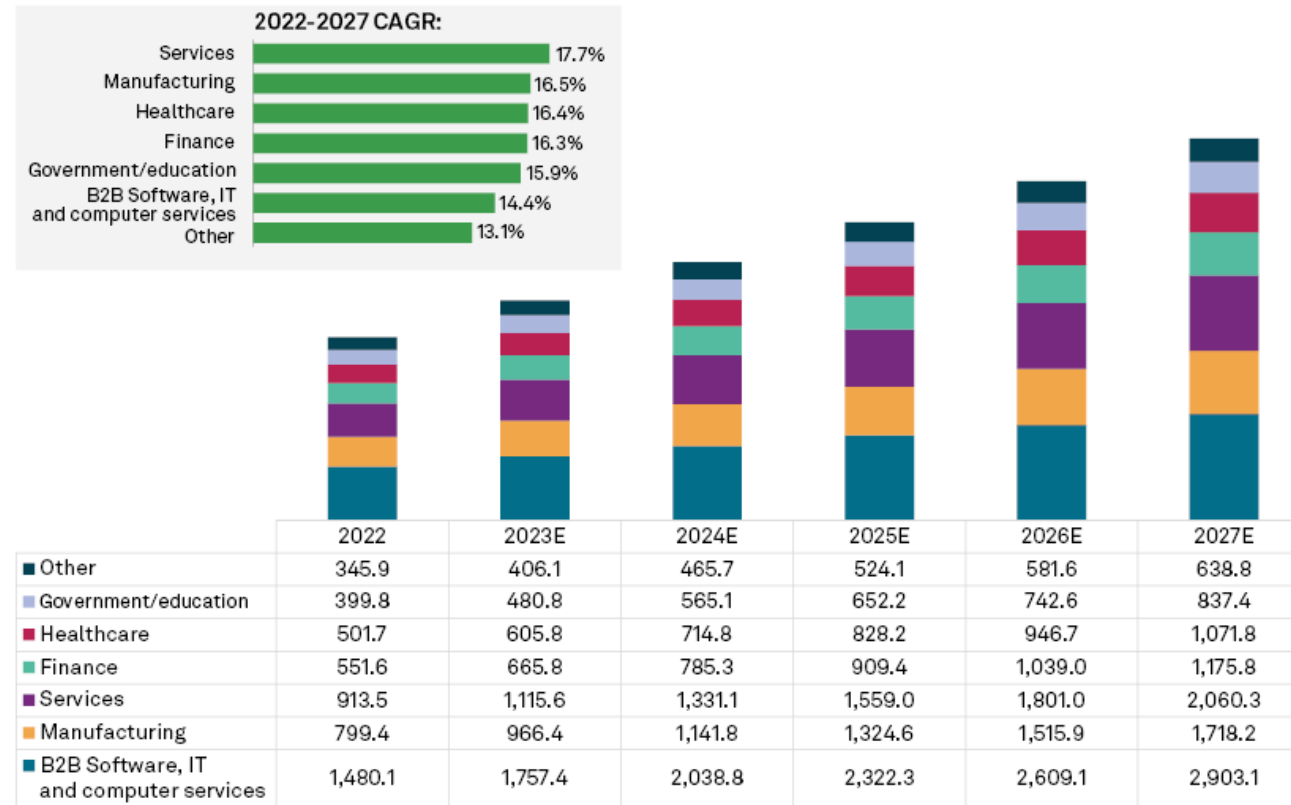


# Contêineres & K8s: Poliglota, Multi-Cloud e Multi-Plataforma



# De uma ferramenta emergente a uma ampla adoção empresarial

Application container ecosystem CAGR and revenue by vertical market (\$M)



CAGR = compound annual growth rate; B2B = business to business; E = estimates.  
Source: Applied Infrastructure & DevOps Market Monitor: Application Container Ecosystem, Q1 2023.  
© 2023 S&P Global.



# Os 5 ABCs da Vida\*

- Seja sempre **C**urioso: ouça com atenção, faça perguntas, maravilhe-se como uma criança, vagueie e pondere, aprenda com os Primeiros Princípios, entre na toca do coelho, leia muito, especialmente sobre assuntos que são novos para você, e leia apenas pela pura alegria de ler e aprender!
- Esteja sempre **C**onectando (os pontos): a curiosidade gera muitos pontos; encontre maneiras de conectá-los, reconheça padrões, adote modelos mentais que ajudem você a conectar os pontos.
- Esteja sempre **C**olaborando: encontre pessoas curiosas, converse com elas, entreviste-as, conecte-se e colabore com elas. A colaboração é a chave para a verdadeira inovação.
- Esteja sempre **C**riando: crie algo a partir do que você aprendeu, seja uma demonstração, um protótipo, uma empresa (!), ou uma postagem no blog/LinkedIn. Crie algo mesmo que você seja o único a vê-lo. Escrever/criar é uma das melhores maneiras de aprender. Leia um livro para buscar inspiração.
- Seja sempre **A**tencioso: nas palavras de Paul Maritz, "Doar > Receber", e nas de Adam Grant em "Givers" - a vida não é um jogo de soma zero. Procure maneiras de ajudar alguém; ensinar alguém é também a melhor maneira de aprender. No mínimo, apenas diga apenas obrigado; assim se chega longe.

\* Source: "The 5 ABCs of Life", Raghvender Arni