Nome do vídeo: T-GCPFCI-B_5_l1_Containers na nuvem

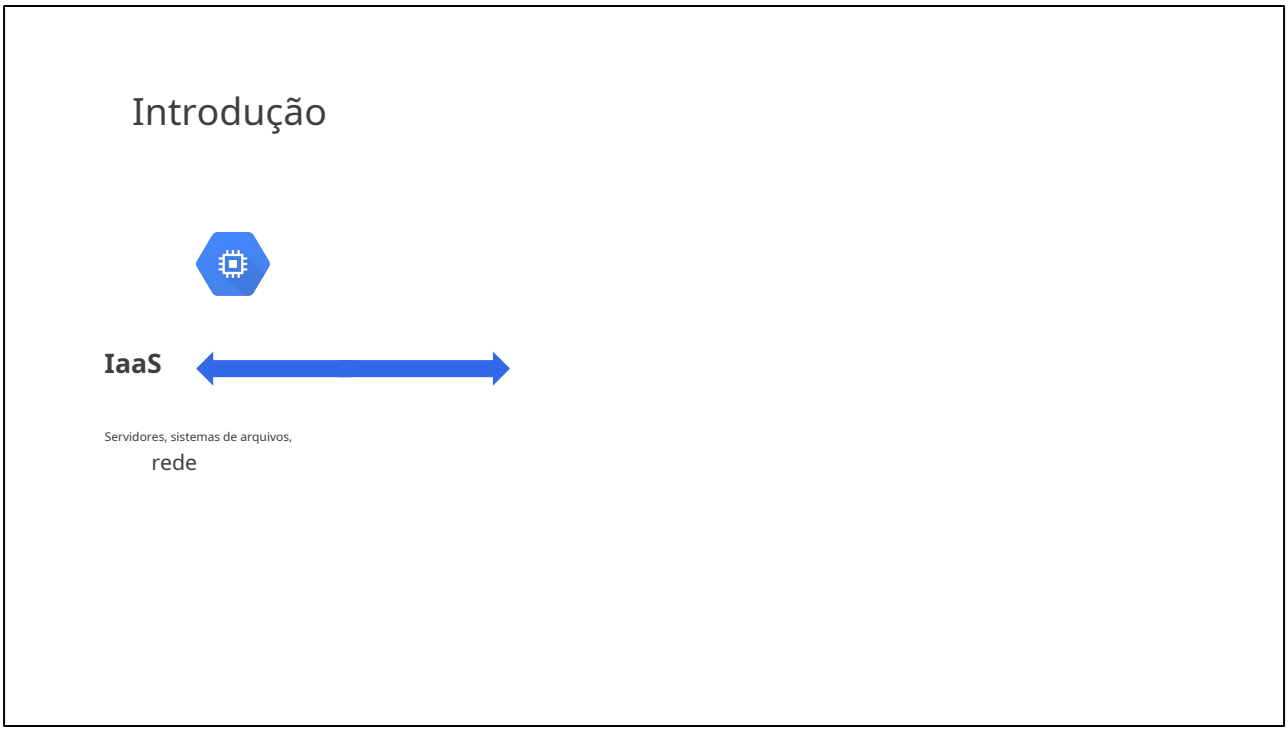
Tipo de conteúdo: Vídeo - Apresentador de palestra

Apresentador: Jim Rambo

Contêineres na Nuvem

Jim Rambo

Bem-vindo a este módulo sobre contêineres e Google Kubernetes Engine.



Nós já discutimos**Compute Engine**, que é o GCP**Infraestrutura como um serviço** oferecendo, com acesso a servidores, sistemas de arquivos e redes.

E**App Engine**que é o GCP**PaaS**oferta.

Agora vou apresentar os contêineres e o Kubernetes Engine, que é um híbrido que fica conceitualmente entre os dois e se beneficia de ambos.

Agenda

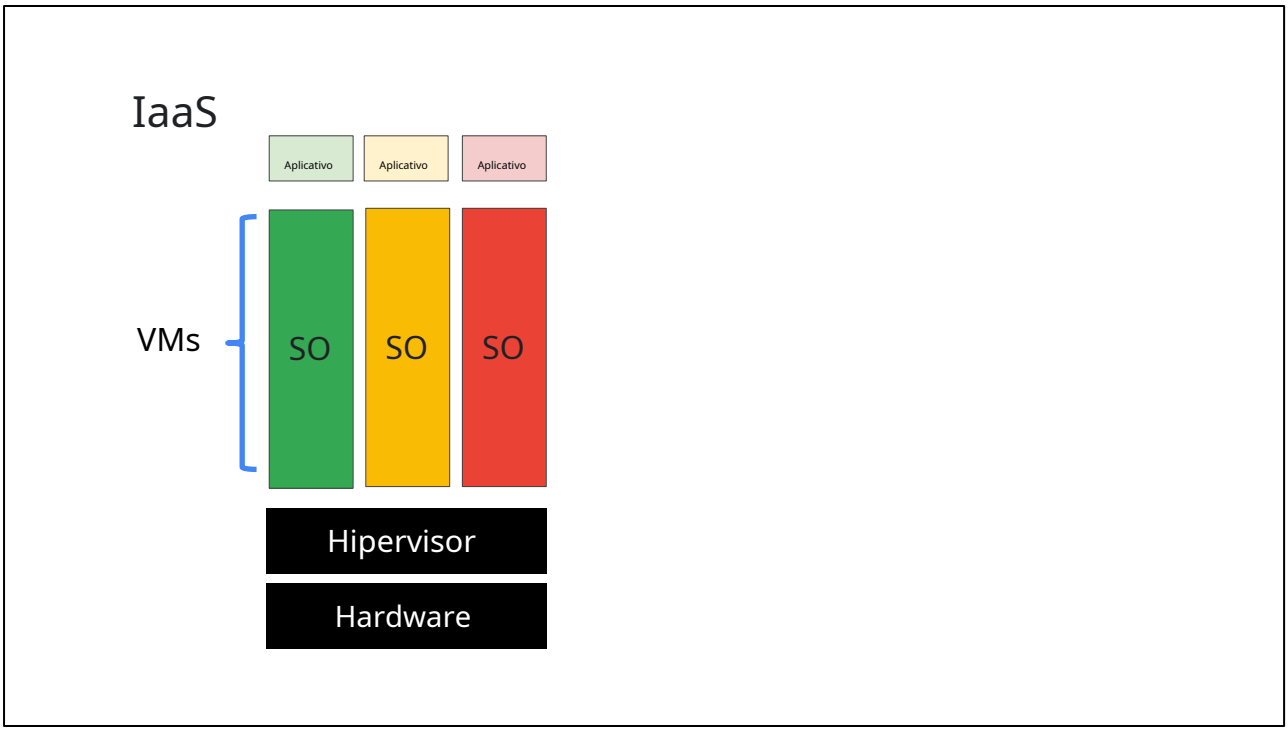
Containers

Kubernetes

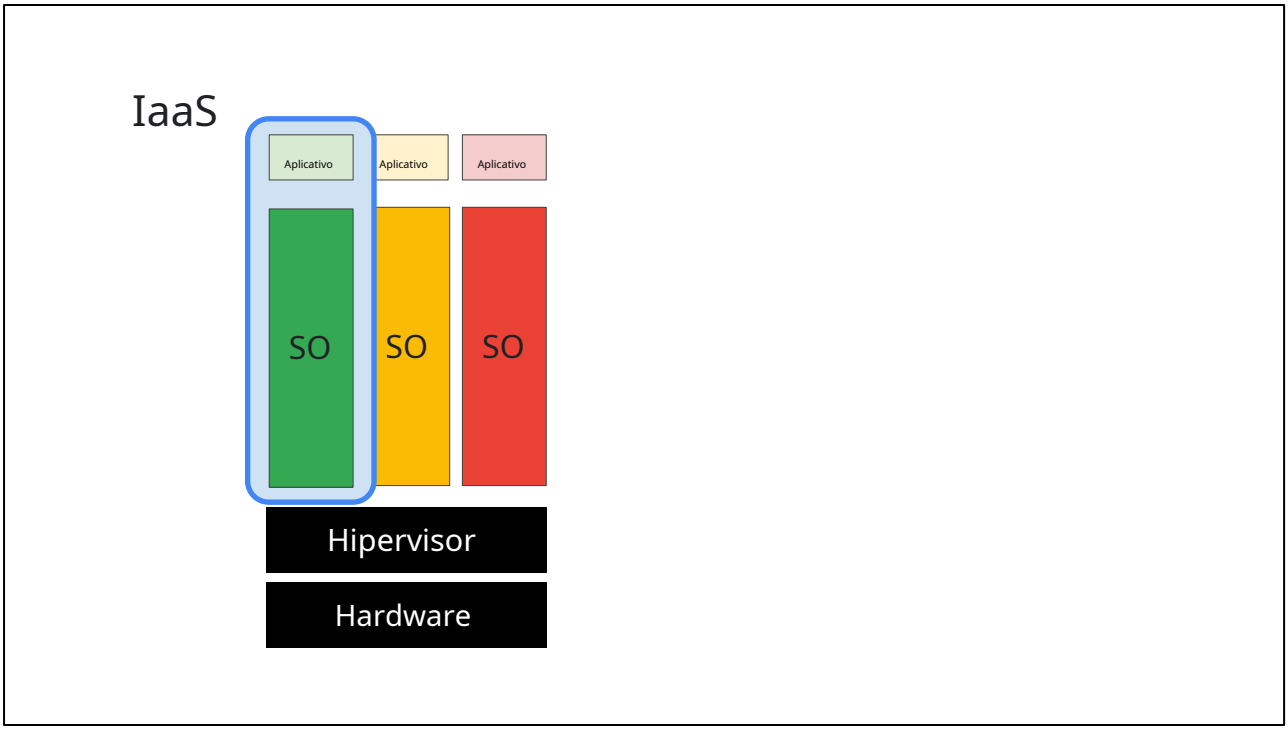
Kubernetes Engine

Laboratório

Vou descrever por que você deseja usar contêineres e como gerenciá-los em**Kubernetes Engine**.

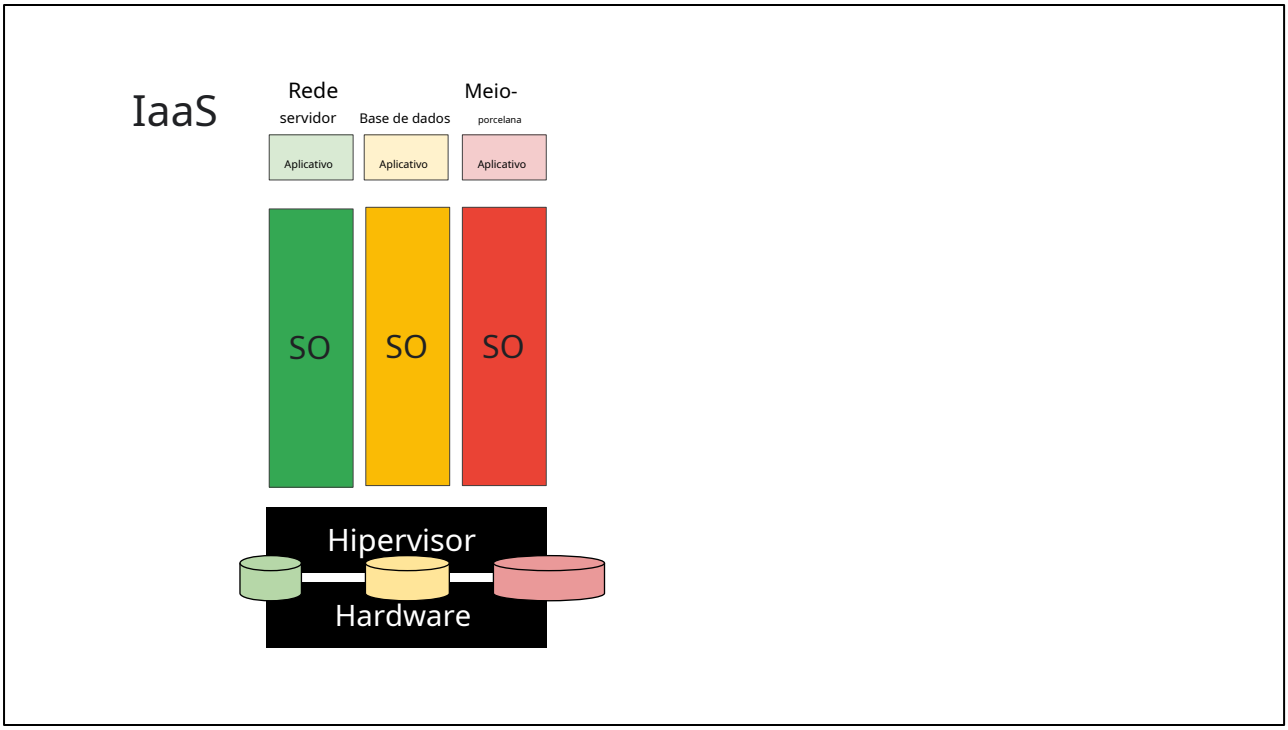


Vamos começar, lembrando que**Infraestrutura como um serviço**permite compartilhar recursos de computação com outros desenvolvedores**virtualizando o hardware**usando**máquinas virtuais**. Cada desenvolvedor pode implantar seu próprio sistema operacional, acessar o hardware e construir seus aplicativos em um ambiente independente com acesso à RAM, sistemas de arquivos, interfaces de rede e assim por diante.

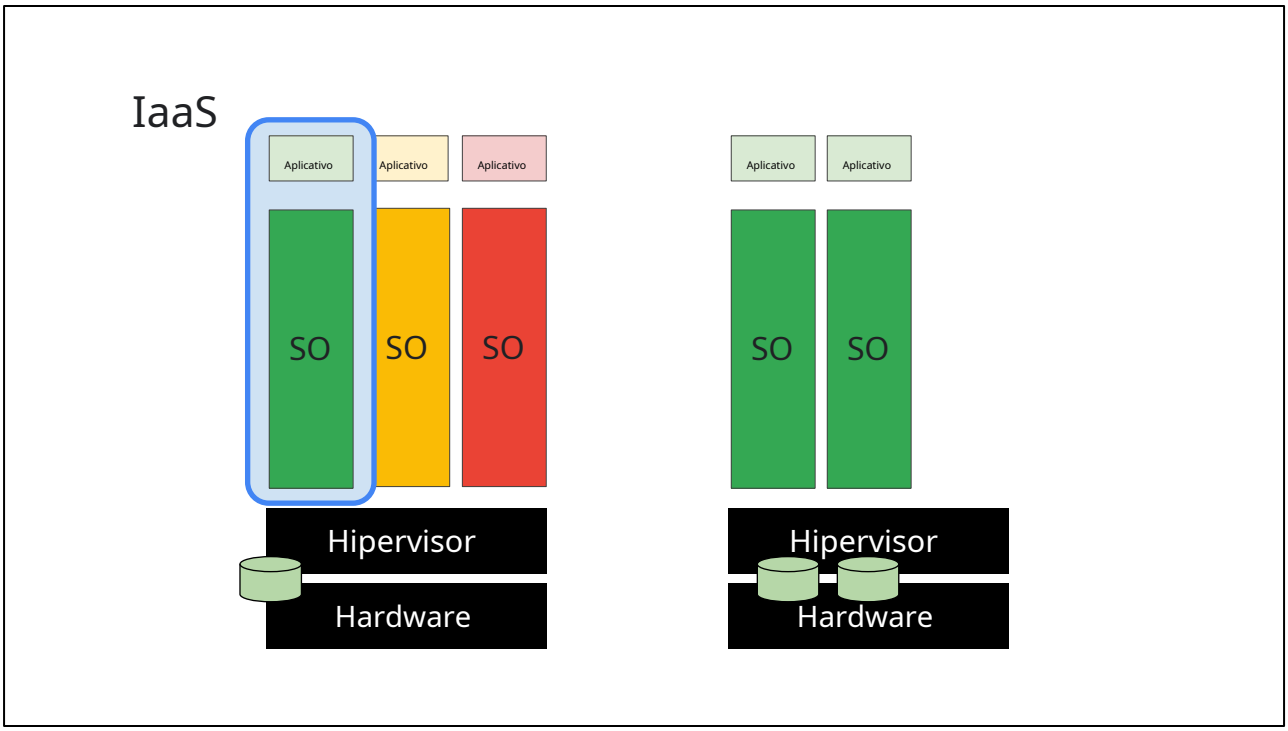


Mas a flexibilidade vem com um custo. A menor unidade de computação é um aplicativo com seu**VM**. O sistema operacional convidado pode ser grande, até gigabytes de tamanho, e leva alguns minutos para inicializar.

1:00

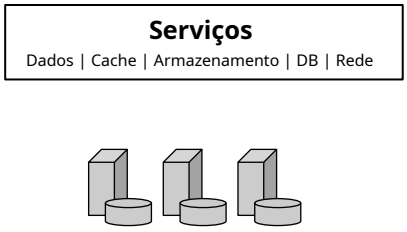


Mas você tem suas ferramentas de escolha em um sistema configurável. Assim, você pode instalar seu runtime, servidor web, banco de dados ou middleware favorito, configurar os recursos do sistema subjacentes, como espaço em disco, E/S em disco ou rede e construir como desejar.

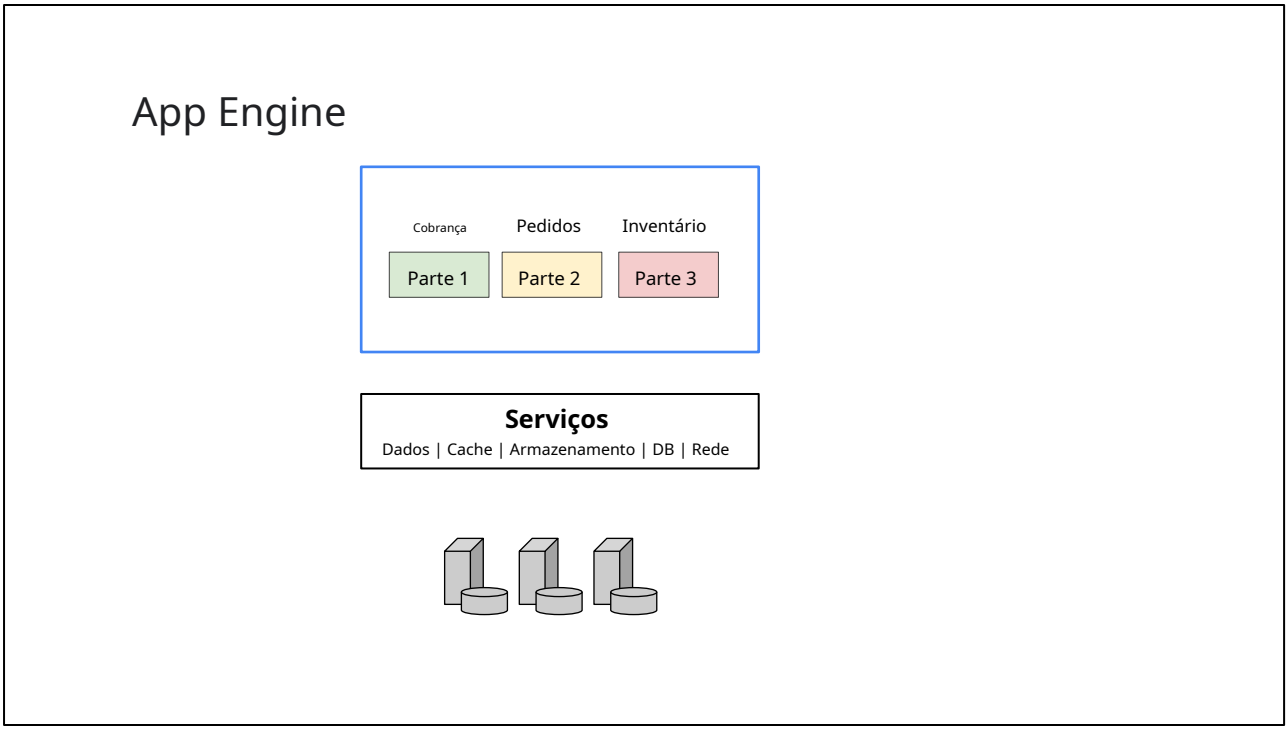


No entanto, à medida que a demanda por seu aplicativo aumenta, você precisa copiar uma VM inteira e inicializar o sistema operacional convidado para cada instância de seu aplicativo, o que pode ser lento e caro.

App Engine



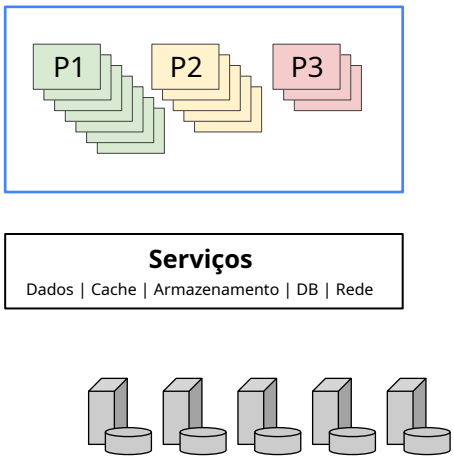
Com**App Engine**você obtém acesso a serviços de programação.



Então tudo que você faz é escrever seu código em**cargas de trabalho**que usam esses serviços e incluem quaisquer bibliotecas dependentes.

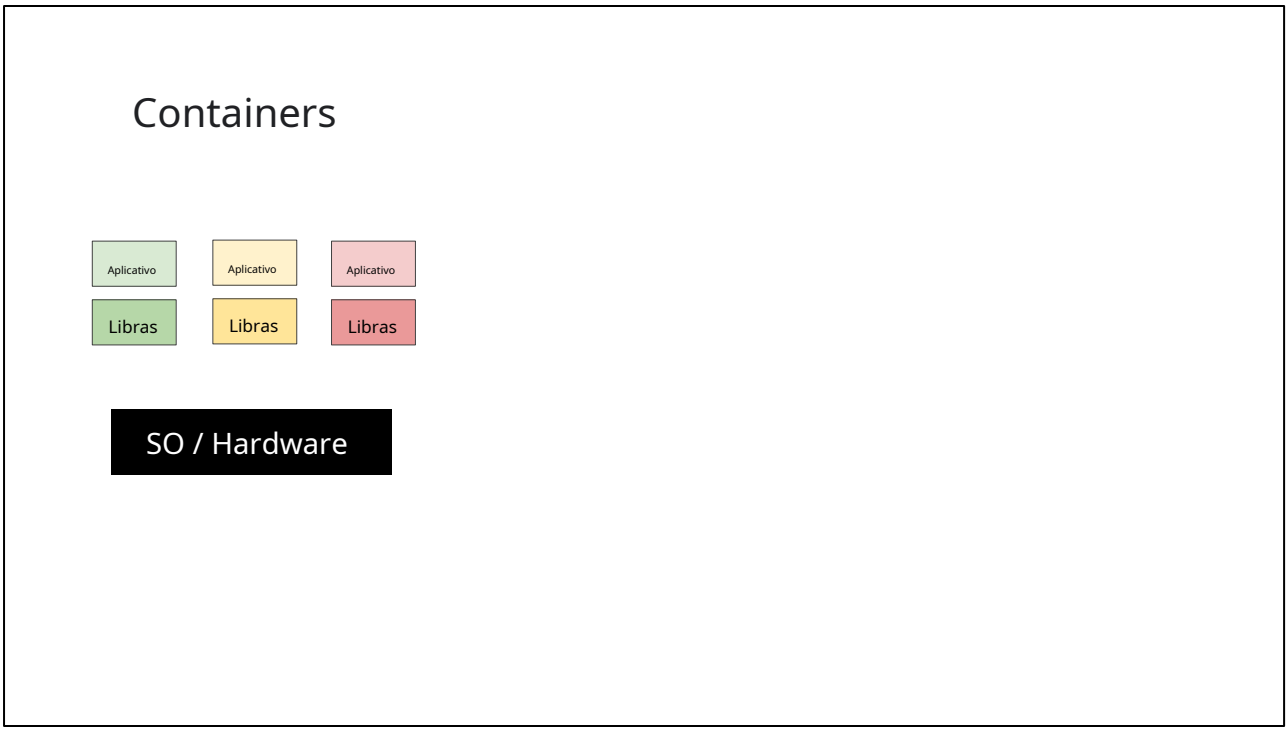
2:00

App Engine



À medida que a demanda por seu aplicativo aumenta, a plataforma dimensiona seu aplicativo de forma contínua e independente por carga de trabalho e infraestrutura.

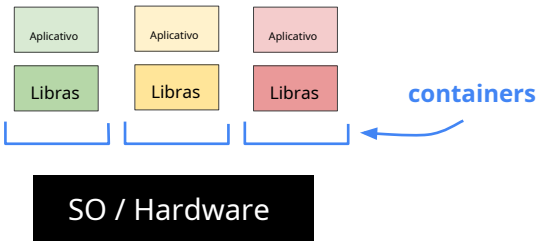
Isso é dimensionado rapidamente, mas você não poderá ajustar a arquitetura subjacente para economizar custos.



É aí que entram os contêineres.

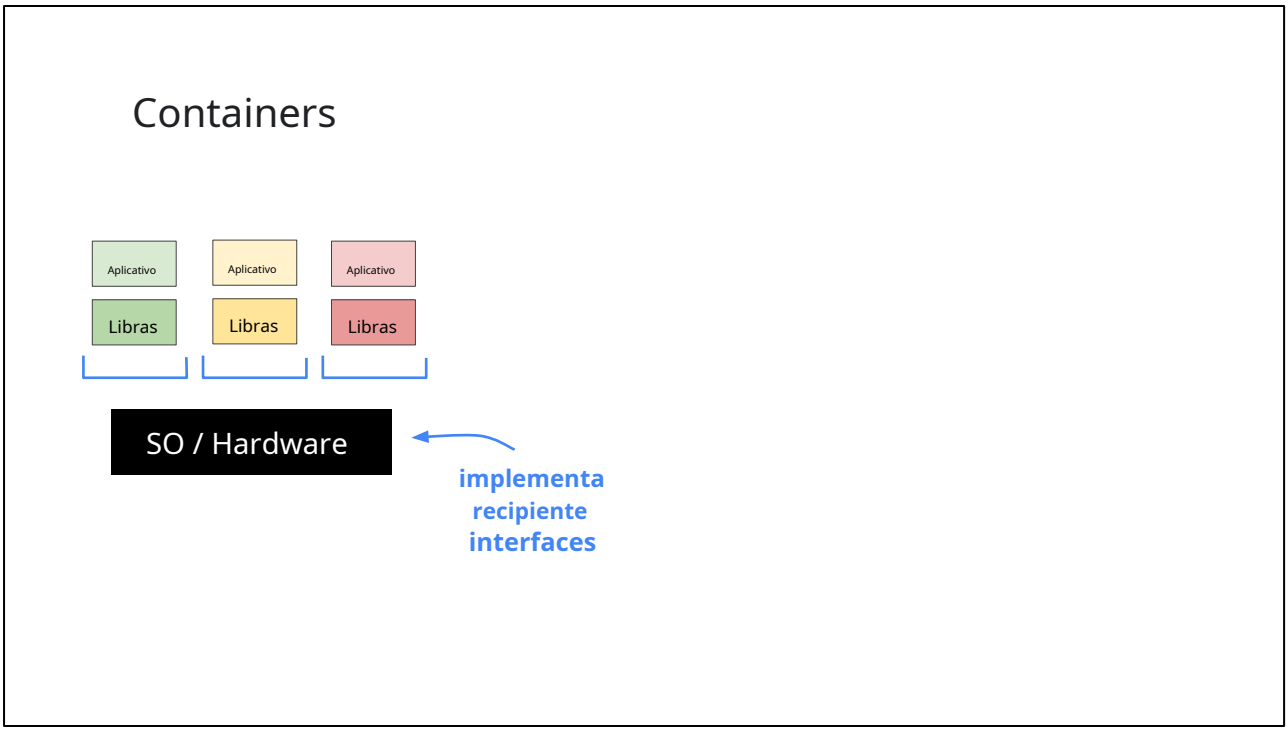
A ideia de um**recipiente**é fornecer a você a escalabilidade independente de cargas de trabalho em PaaS e uma camada de abstração do SO e hardware em IaaS.

Containers



O que você obtém é um**caixa invisível**em torno de seu código e suas dependências, com acesso limitado ao seu próprio**partição**do sistema de arquivos e hardware.

Ele requer apenas algumas chamadas de sistema para criar e é iniciado tão rapidamente quanto um processo.

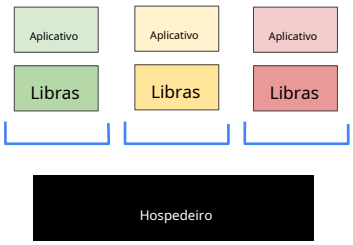


Tudo o que você precisa em cada host é um kernel do sistema operacional que suporte contêineres e um tempo de execução de contêiner.

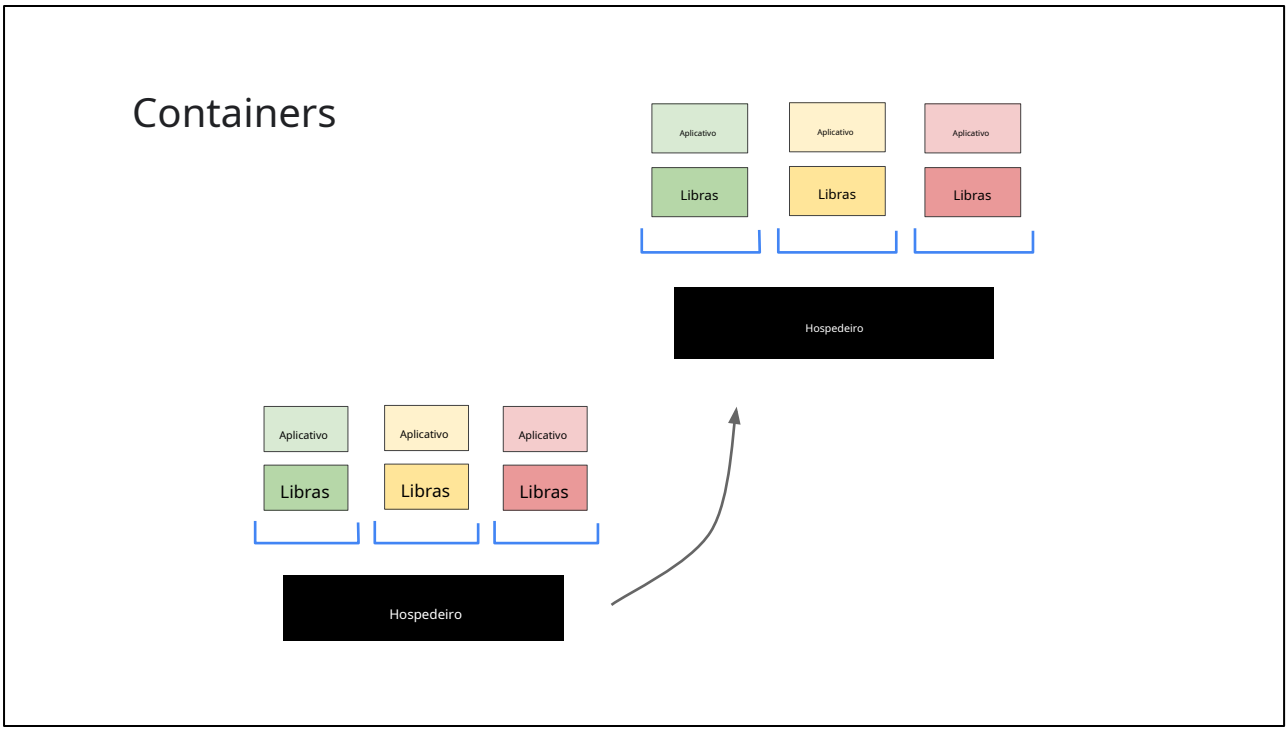
Em essência, você é**virtualizando o SO**. Ele é dimensionado como o PaaS, mas oferece quase a mesma flexibilidade que o IaaS.

2:15

Containers

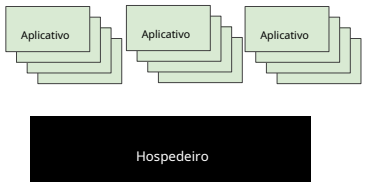


Com essa abstração, seu código é ultraportátil e você pode tratar o sistema operacional e o hardware como uma caixa preta.



Assim, você pode ir do desenvolvimento à preparação, à produção ou do seu laptop à nuvem, sem alterar ou reconstruir nada.

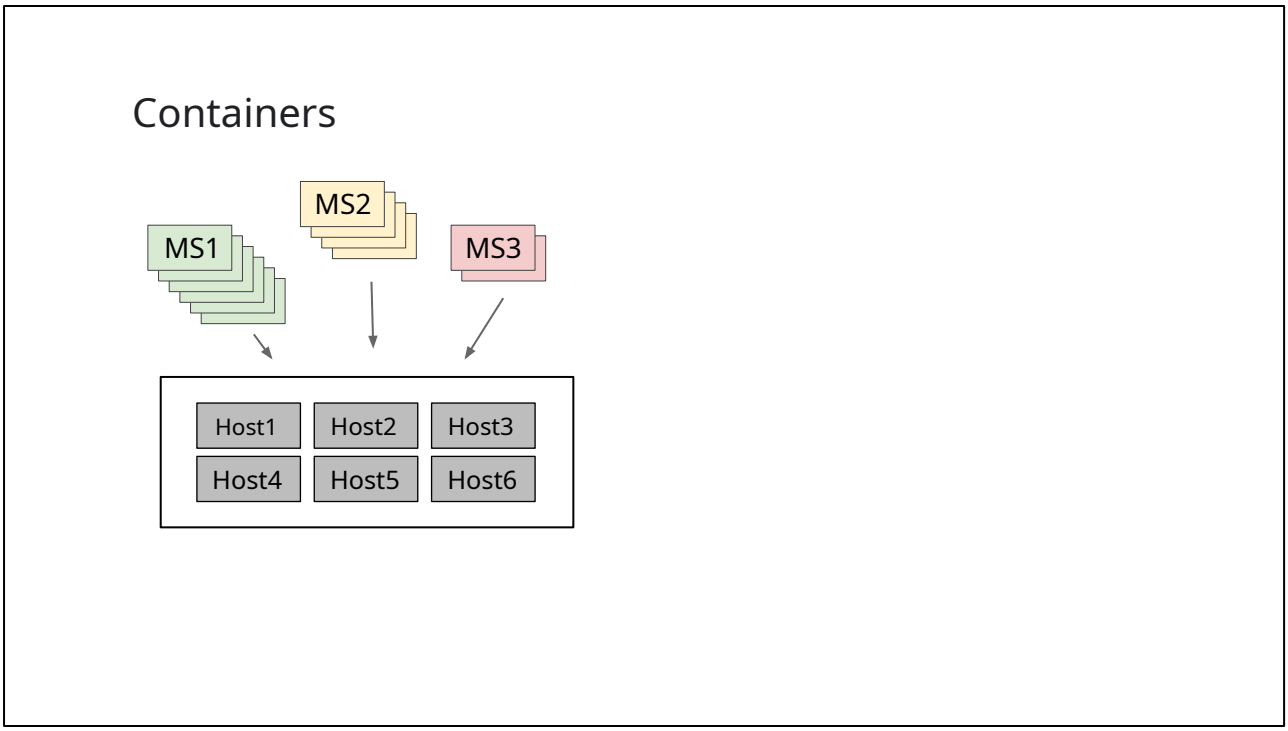
Containers



Se você deseja dimensionar, por exemplo, um servidor web, pode fazê-lo em segundos e implantar dezenas ou centenas deles (dependendo do tamanho ou da carga de trabalho) em um único host.

Agora, esse é um exemplo simples de dimensionamento de um contêiner executando todo o aplicativo em um único host.

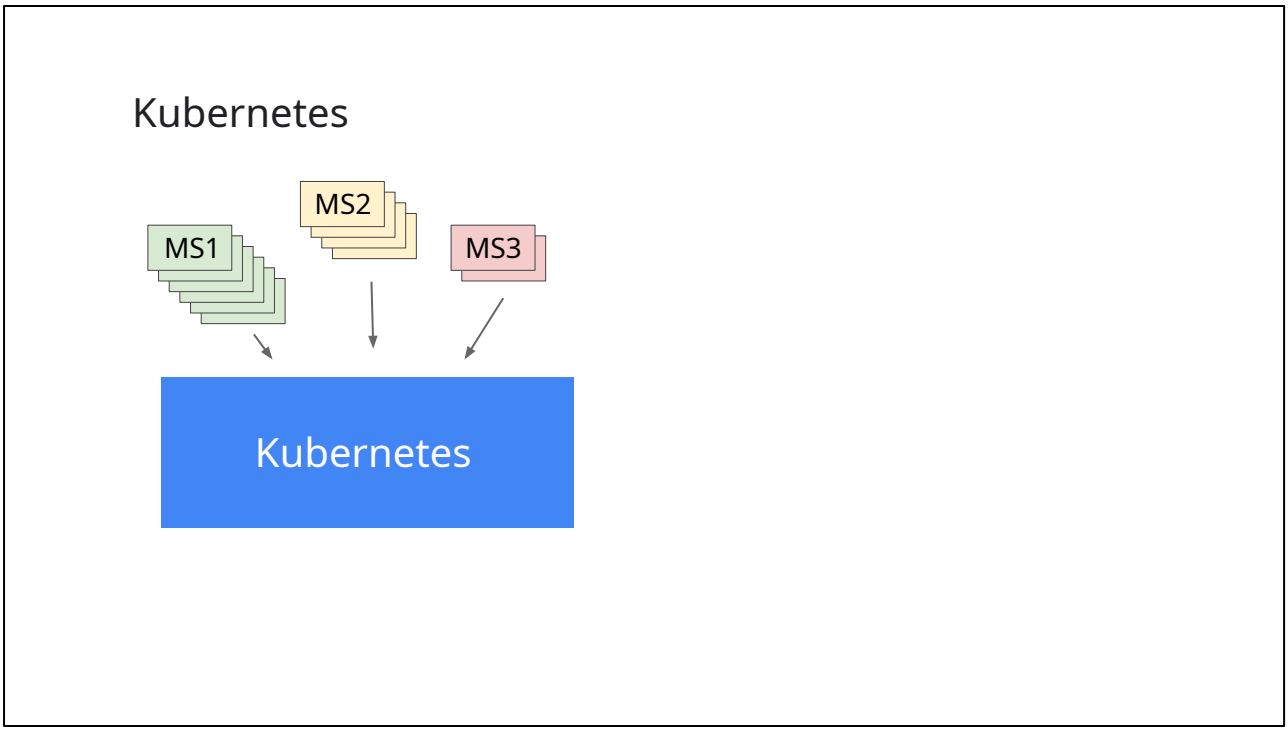
2:45



Você provavelmente desejará construir seus aplicativos usando muitos contêineres, cada um executando sua própria função, como**microsserviços**.

Se você os construir dessa maneira e conectá-los a conexões de rede, poderá torná-los modulares, implantar facilmente e dimensionar independentemente em um grupo de**anfitriões**.

E os hosts podem escalar para cima e para baixo e iniciar e parar contêineres conforme a demanda do seu aplicativo muda ou quando os hosts falham.



Uma ferramenta que ajuda você a fazer isso bem é**Kubernetes**.

Kubernetesfacilita a orquestração de muitos contêineres em muitos hosts, dimensioná-los como microsserviços e implantar implementações e reversões.

Primeiro, mostrarei como você cria e executa contêineres.

Vou usar uma ferramenta de código aberto chamada**Janela de encaixe**que define um formato para agrupar seu aplicativo, suas dependências e configurações específicas da máquina em um contêiner; você pode usar uma ferramenta diferente como**Construtor de contêineres do Google**. Você decide.

3:30

app.py

```
from flask import Flask app
= Flask(__name__)

@app.route("/")
def olá():
    return "Olá Mundo!\n"

@app.route("/versão")
versão def():
    return "Helloworld 1.0\n"

if __name__ == "__main__":
    app.run(host='0.0.0.0')
```

Aqui está um exemplo de algum código que você pode ter escrito.

app.py

```
from flask import Flask app
= Flask(__name__)

@app.route("/")
def olá():
    return "Olá Mundo!\n"

@app.route("/versão")
versão def():
    return "Helloworld 1.0\n"

if __name__ == "__main__":
    app.run(host='0.0.0.0')
```

É um aplicativo Python.

app.py

```
from flask import Flask app
= Flask(__name__)

@app.route("/")
def olá():
    return "Olá Mundo!\n"

@app.route("/versão")
versão def():
    return "Helloworld 1.0\n"

if __name__ == "__main__":
    app.run(host='0.0.0.0')
```

Diz "Olá Mundo"

app.py

```
from flask import Flask app
= Flask(__name__)

@app.route("/")
def olá():
    return "Olá Mundo!\n"

@app.route("/versão")
versão def():
    return "Helloworld 1.0\n"

if __name__ == "__main__":
    app.run(host='0.0.0.0')
```

Ou se você atingir esse ponto final,

app.py

```
from flask import Flask app
= Flask(__name__)

@app.route("/")
def olá():
    return "Olá Mundo!\n"

@app.route("/versão")
versão def():
    return "Helloworld 1.0\n"

if __name__ == "__main__":
    app.run(host='0.0.0.0')
```

dá-lhe a versão.

Então, como você coloca esse aplicativo no Kubernetes?

Você tem que pensar na sua versão do Python, qual dependência você tem do Flask,

3:45

requisitos.txt

```
Frasco==0,12
uwsgi==2.0.15
```

como usar o arquivo requirements.txt, como instalar o Python e assim por diante.

Dockerfile

```
DO Ubuntu: 18.10
RUN apt-get update -y && \
        apt-get install -y python3-pip python3-dev COPIAR
requirements.txt /app/requirements.txt WORKDIR /app

EXECUTAR pip3 install -r requirements.txt
COPY . /aplicativo
ENDPOINT ["python3", "app.py"]
```

Então você usa um**Dockerfile**para especificar como seu código é empacotado em um contêiner.

Por exemplo, se você é um desenvolvedor e está acostumado a usar o Ubuntu com todas as suas ferramentas, comece por aí.

Dockerfile

```
DO Ubuntu: 18.10
RUN apt-get update -y && \
        apt-get install -y python3-pip python3-dev COPIAR
requirements.txt /app/requirements.txt WORKDIR /app

EXECUTAR pip3 install -r requirements.txt
COPY . /aplicativo
ENDPOINT ["python3", "app.py"]
```

Você pode instalar o Python da mesma maneira que faria em seu ambiente de desenvolvimento.

Dockerfile

```
DO Ubuntu: 18.10
RUN apt-get update -y && \
        apt-get install -y python3-pip python3-dev COPIAR
requirements.txt /app/requirements.txt WORKDIR /app

EXECUTAR pip3 install -r requirements.txt
COPY . /aplicativo
ENDPOINT ["python3", "app.py"]
```

Você pode pegar esse arquivo de requisitos do Python que você conhece.

Dockerfile

```
DO Ubuntu: 18.10
RUN apt-get update -y && \
        apt-get install -y python3-pip python3-dev COPIAR
requirements.txt /app/requirements.txt WORKDIR /app

EXECUTAR pip3 install -r requirements.txt
COPY . /aplicativo
ENDPOINT ["python3", "app.py"]
```

E você pode usar ferramentas dentro**Janela de encaixe**ou**Construtor de contêineres**para instalar suas dependências do jeito que você quiser.

Dockerfile

```
DO Ubuntu: 18.10
RUN apt-get update -y && \
        apt-get install -y python3-pip python3-dev COPIAR
requirements.txt /app/requirements.txt WORKDIR /app

EXECUTAR pip3 install -r requirements.txt
COPY . /aplicativo
ENDPOINT ["python3", "app.py"]
```

Eventualmente, ele produz um aplicativo,

Dockerfile

```
DO Ubuntu: 18.10
RUN apt-get update -y && \
        apt-get install -y python3-pip python3-dev COPIAR
requirements.txt /app/requirements.txt WORKDIR /app

EXECUTAR pip3 install -r requirements.txt
COPY . /aplicativo
ENDPOINT ["python3", "app.py"]
```

e aqui está como você executá-lo.

Construir e executar

```
$> docker build -t py-server . $>
docker run -d py-server
```

Então você usa o "**compilação do docker**" comando para construir o contêiner.

Isso cria o contêiner e o armazena localmente como um executável**imagem**. Você pode salvar e carregar a imagem em um serviço de registro de contêiner e compartilhá-la ou baixá-la de lá.

Então você usa o "**execução do docker**" comando para executar a imagem.

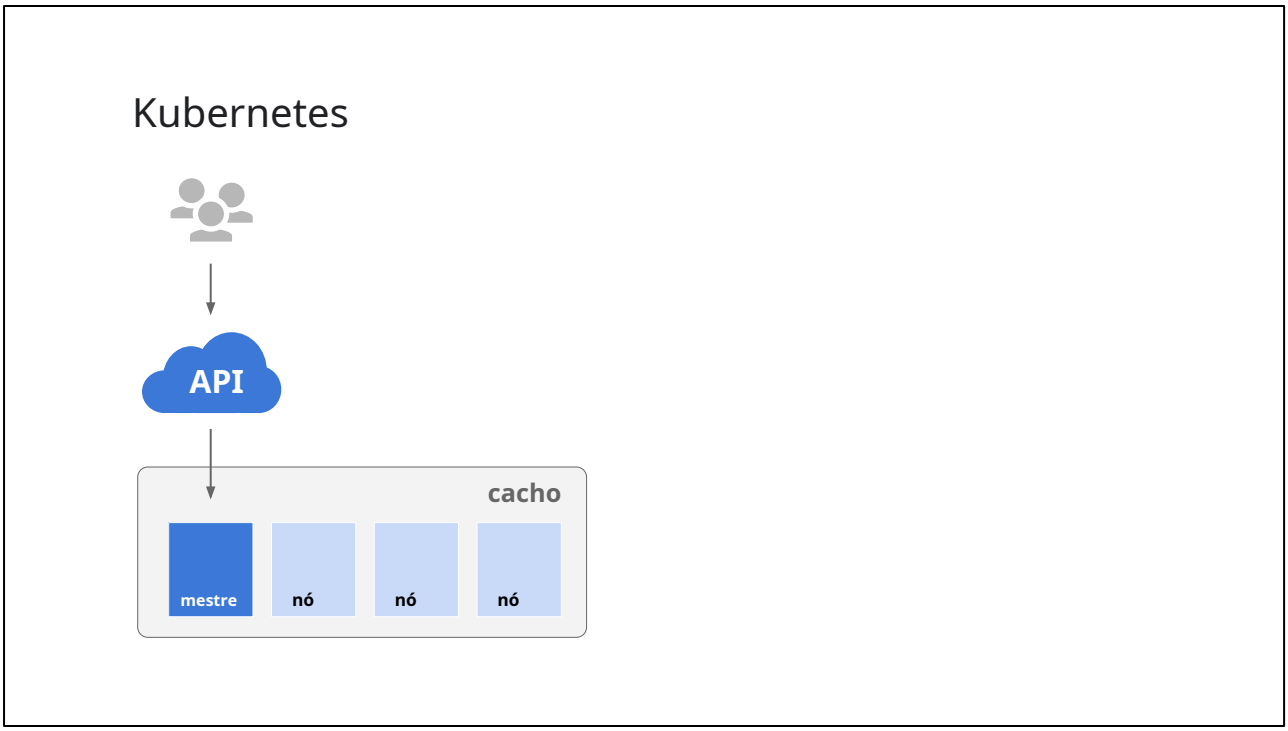
Como se vê, as aplicações de empacotamento são apenas cerca de 5% do problema. O resto tem a ver com: configuração de aplicativos, descoberta de serviços, gerenciamento de atualizações e monitoramento. Esses são os componentes de um sistema distribuído confiável, escalável.

4:45

Agora, eu vou te mostrar onde**Kubernetes**entra.

Kubernetes é um software de código aberto**orquestrador**que abstrai contêineres em um nível mais alto para que você possa gerenciar e dimensionar melhor seus aplicativos.

No nível mais alto, o Kubernetes é um conjunto de APIs que você pode usar para implantar contêineres em um conjunto de**nós**chamado de**cacho**.

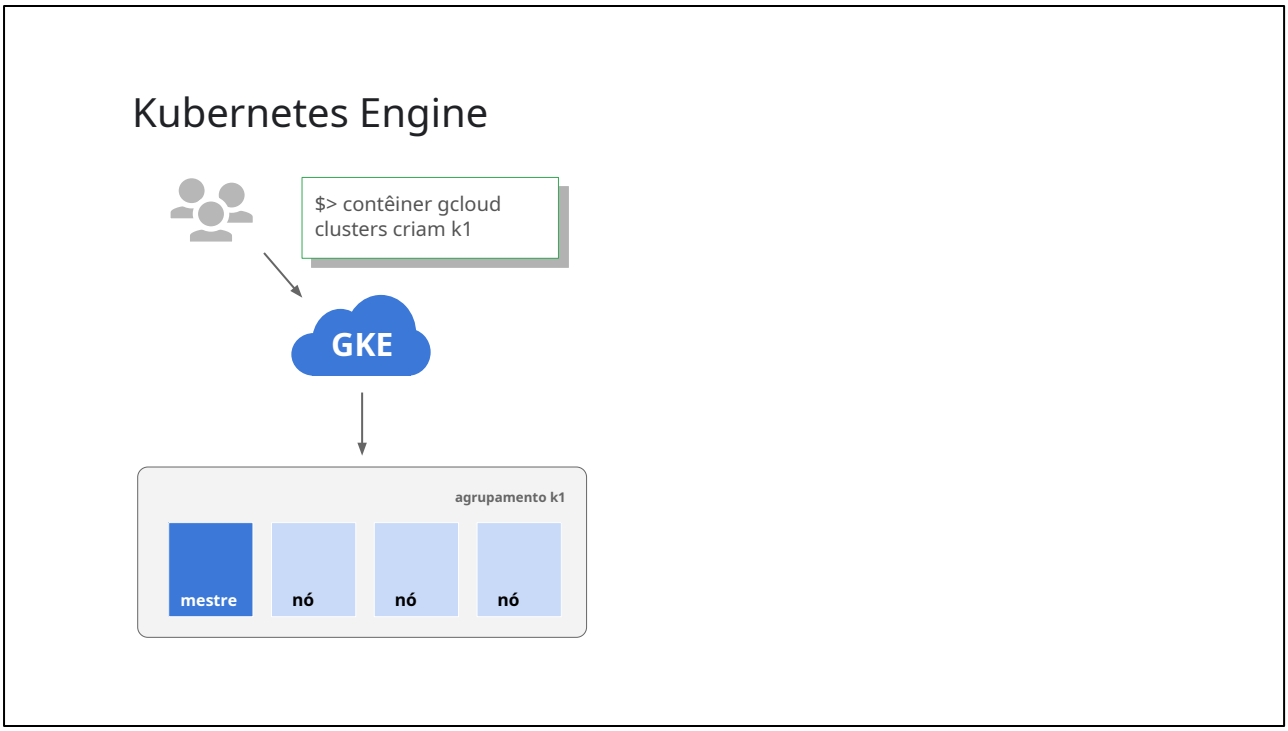


O sistema é dividido em um conjunto de**mestre**componentes que funcionam como o plano de controle e um conjunto de**nós**que executam contêineres. No Kubernetes, um nó representa uma instância de computação, como uma máquina. No Google Cloud, os nós são máquinas virtuais executadas no Compute Engine.

Você pode descrever um conjunto de aplicativos e como eles devem interagir entre si e o Kubernetes descobre como fazer isso acontecer

Agora que você criou um contêiner, você desejará implantar um em um**cacho**.

5:30



O Kubernetes pode ser configurado com muitas opções e complementos, mas pode ser demorado para inicializar do zero. Em vez disso, você pode inicializar o Kubernetes usando **Kubernetes Engine**ou (GKE).

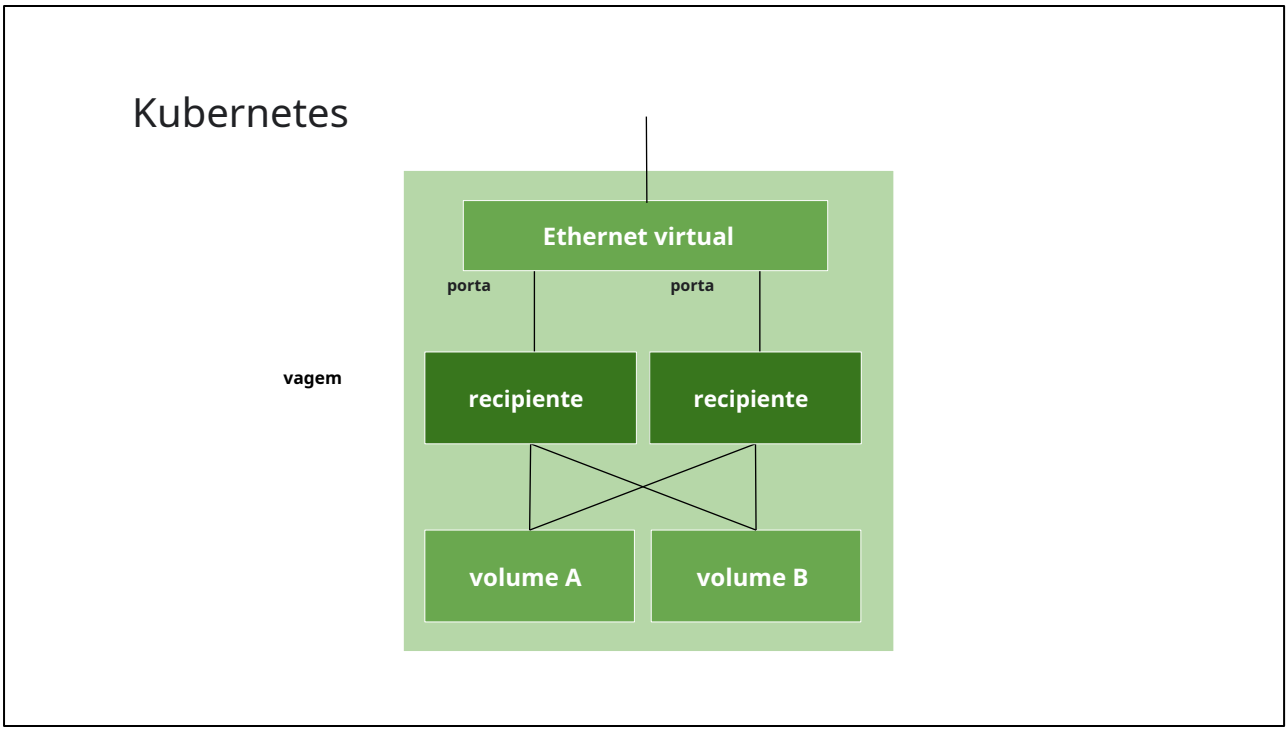
GKEé um Kubernetes hospedado pelo Google. Os clusters do GKE podem ser personalizados e são compatíveis com diferentes tipos de máquina, número de nós e configurações de rede.

Para iniciar o Kubernetes em um**cacho**no GKE, tudo o que você faz é executar este comando:

Neste ponto, você deve ter um cluster chamado 'k1' configurado e pronto para ser usado.

Você pode verificar seu status no console de administração.

6:00



Então você implanta**containers**em nós usando um wrapper em torno de um ou mais contêineres chamados de Pod.

UMA**Cápsula**é a menor unidade no Kubernetes que você cria ou implanta. Um pod representa um processo em execução em seu cluster como um componente de seu aplicativo ou um aplicativo inteiro.

Geralmente, você tem apenas um contêiner por pod, mas se tiver vários contêineres com uma dependência física, poderá empacotá-los em um único pod e compartilhar rede e armazenamento. O Pod fornece um IP de rede exclusivo e um conjunto de portas para seus contêineres, além de opções que controlam como os contêineres devem ser executados.

Os contêineres dentro de um pod podem se comunicar uns com os outros usando o host local e as portas que permanecem fixas à medida que são iniciadas e interrompidas em nós diferentes.

6:30

Uma maneira de executar um contêiner em um pod no Kubernetes é usar o**kubectl run** comando. Aprenderemos uma maneira melhor mais adiante neste módulo, mas isso ajuda você a começar rapidamente.

Isso inicia um**Desdobramento, desenvolvimento**com um contêiner rodando em um**Cápsula**e o contêiner dentro do Pod é uma imagem do servidor nginx.

UMA**Desdobramento, desenvolvimento**representa um grupo de réplicas do mesmo pod e mantém seus pods em execução mesmo quando os nós executados falham. Pode representar um componente de um aplicativo ou um aplicativo inteiro. Neste caso, é o servidor web nginx.

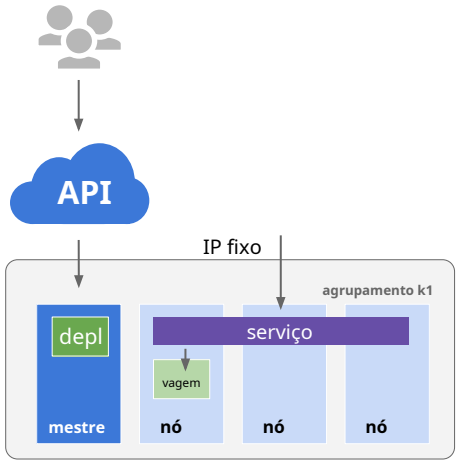
Para ver os pods nginx em execução, execute o comando:

$ kubectl**obter vagens**

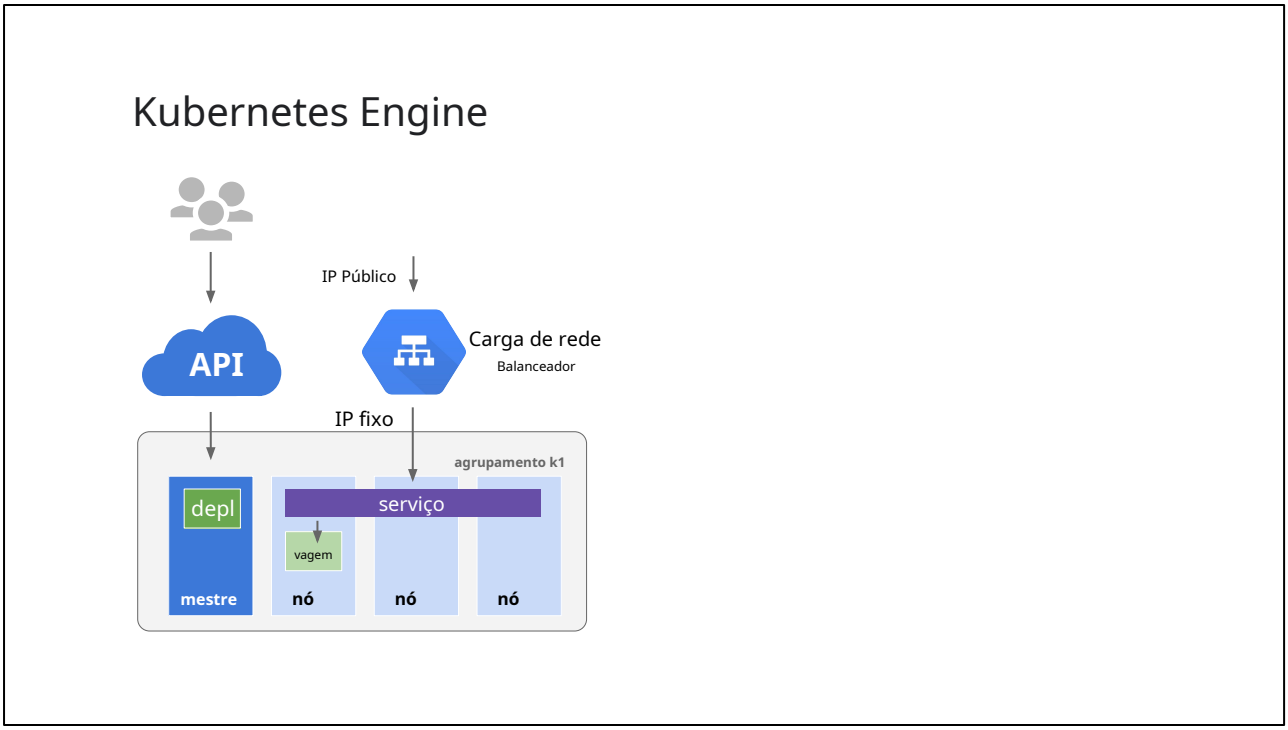
7:00

Por padrão, os pods em uma implantação só podem ser acessados dentro do cluster do GKE. Para torná-los disponíveis publicamente, você pode conectar um balanceador de carga ao seu Deployment executando o**kubectl expor**comando:

Kubernetes Engine



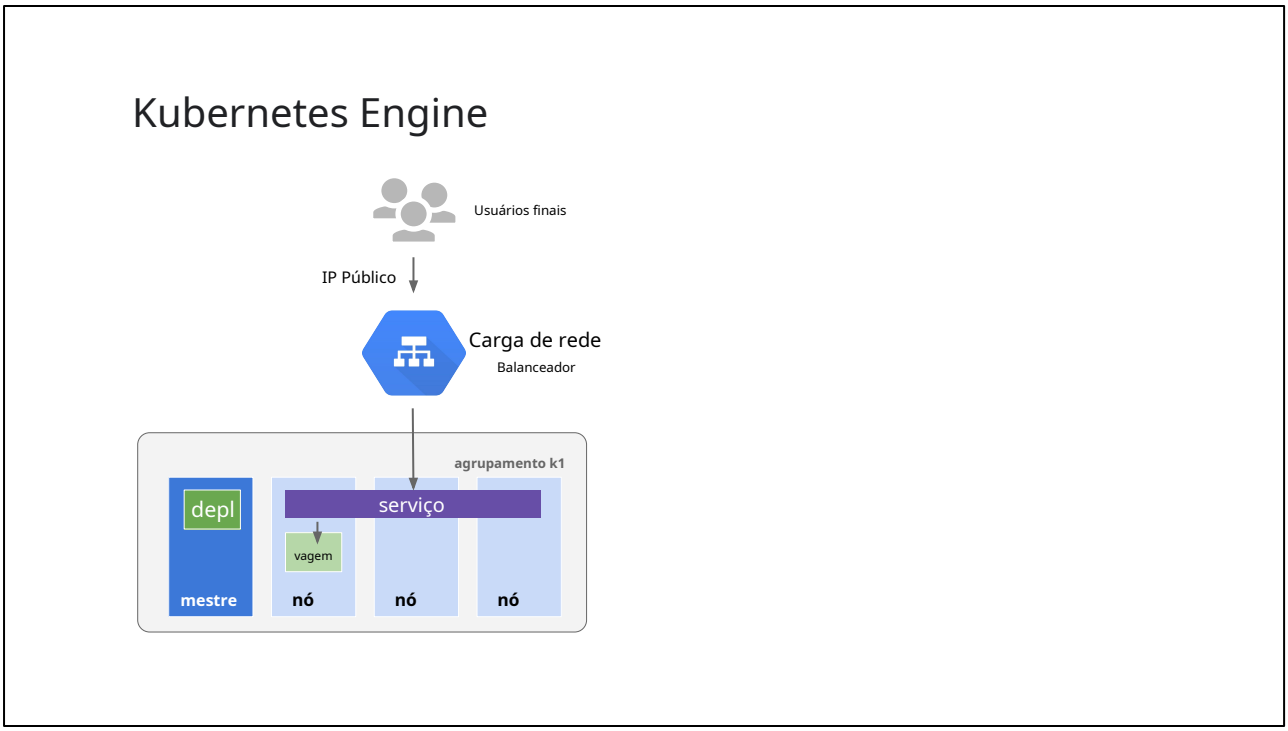
O Kubernetes cria um**Serviço**com um IP fixo para seus Pods,



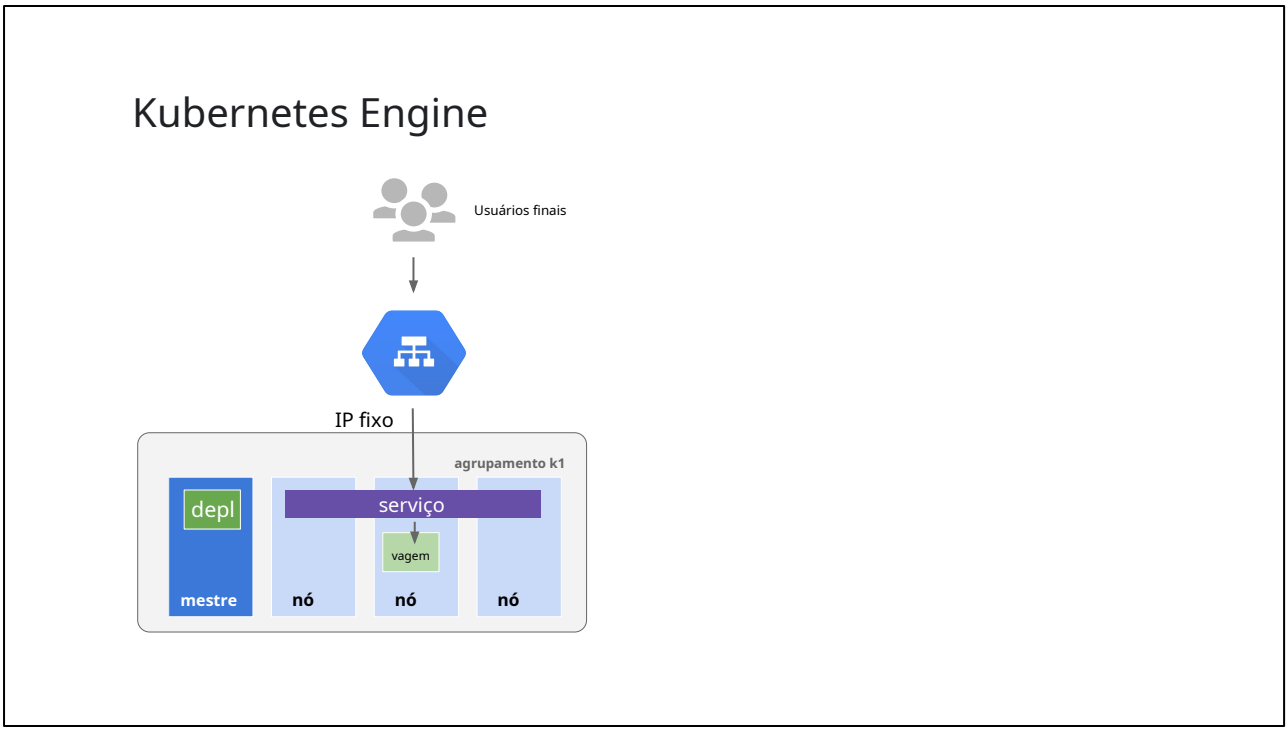
e um controlador diz "Preciso anexar um**balanceamento de carga**r com um endereço IP público para esse**Serviço**para que outros fora do cluster possam acessá-lo".

Dentro**GKE**, o balanceador de carga é criado como um**Balanceador de carga de rede**.

7:30



Qualquer cliente que atingir esse endereço IP será roteado para um Pod por trás do Serviço, neste caso, há apenas um - seu Pod nginx simples.



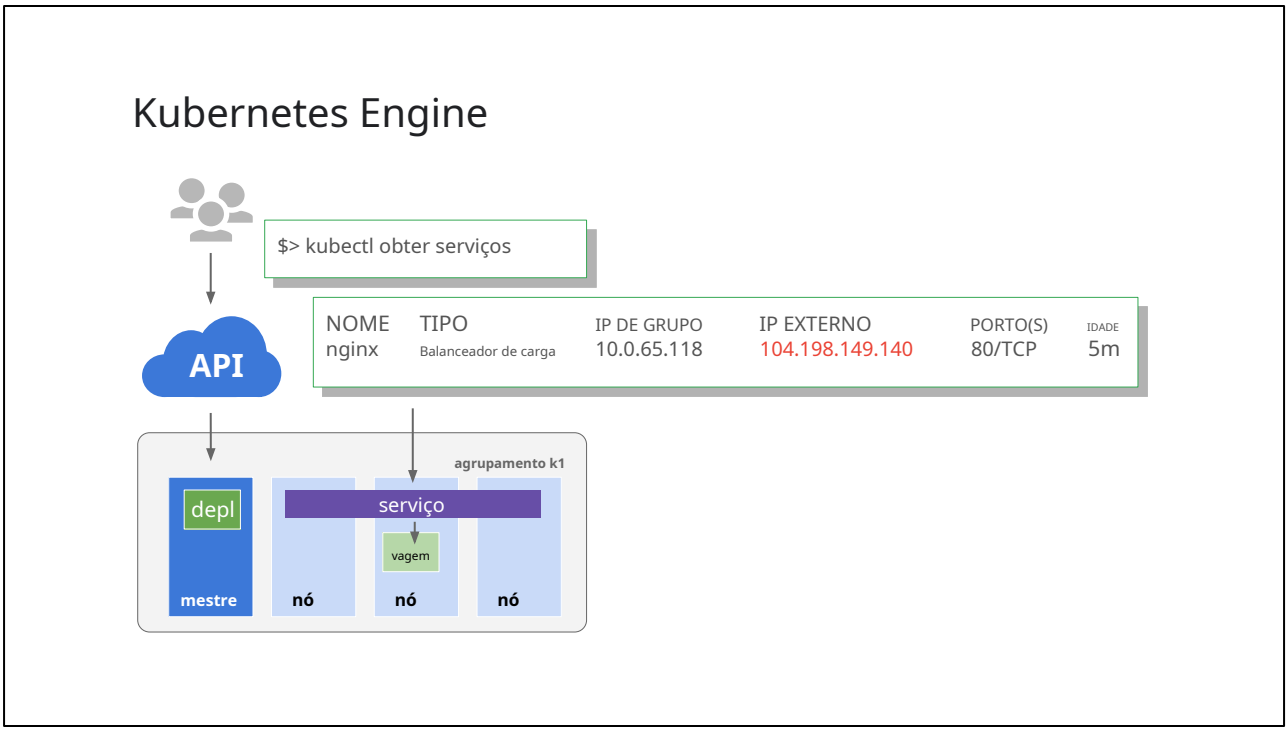
UMA**Serviço**é uma abstração que define um conjunto lógico de pods e uma política para acessá-los.

À medida que as implantações criam e destroem pods, os pods obtêm seu próprio endereço IP. Mas esses endereços não permanecem estáveis ao longo do tempo.

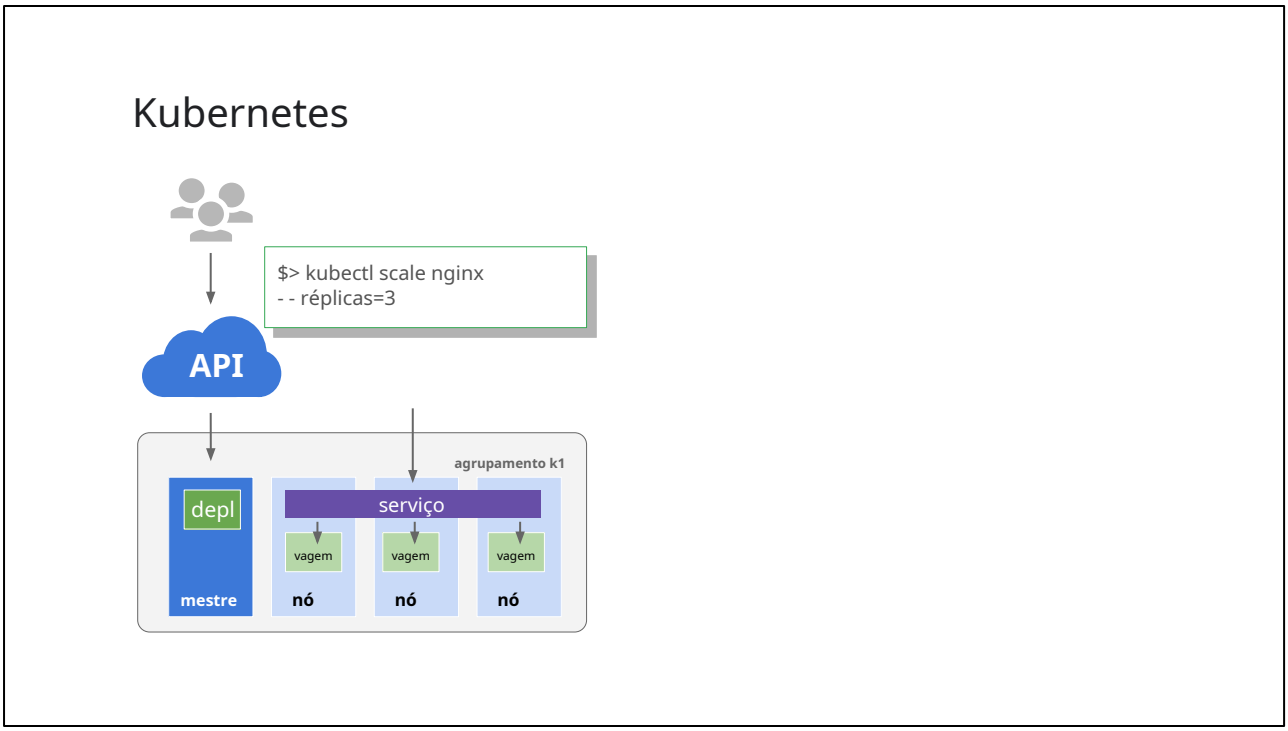
Um serviço agrupa um conjunto de pods e fornece um endpoint estável (ou IP fixo) para eles.

Por exemplo, se você criar dois conjuntos de pods chamados front-end e back-end e colocá-los atrás de seus próprios serviços, os pods de back-end podem mudar, mas os pods de front-end não estão cientes disso. Eles simplesmente se referem ao serviço de back-end.

8:15



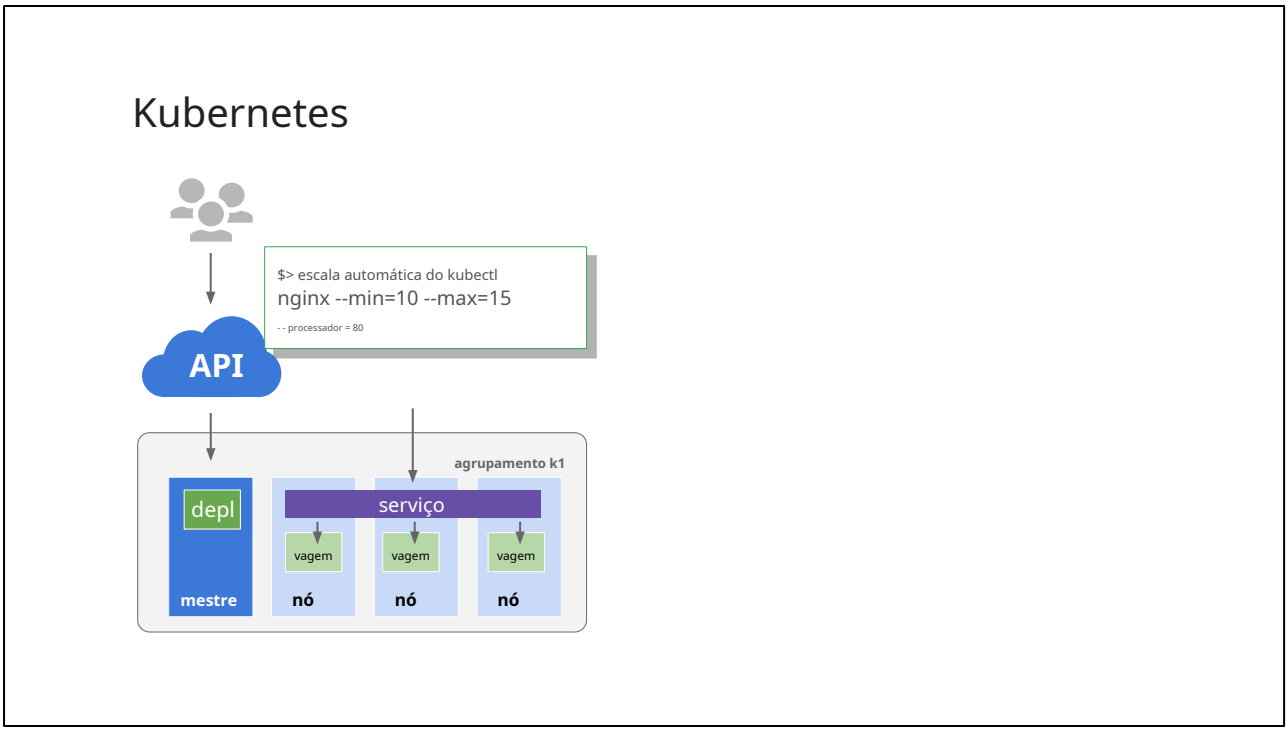
Você pode executar o**kubectl obter serviços**comando para obter o IP público para atingir o contêiner nginx remotamente.



Para dimensionar uma implantação, execute o**escala kubectl**comando.

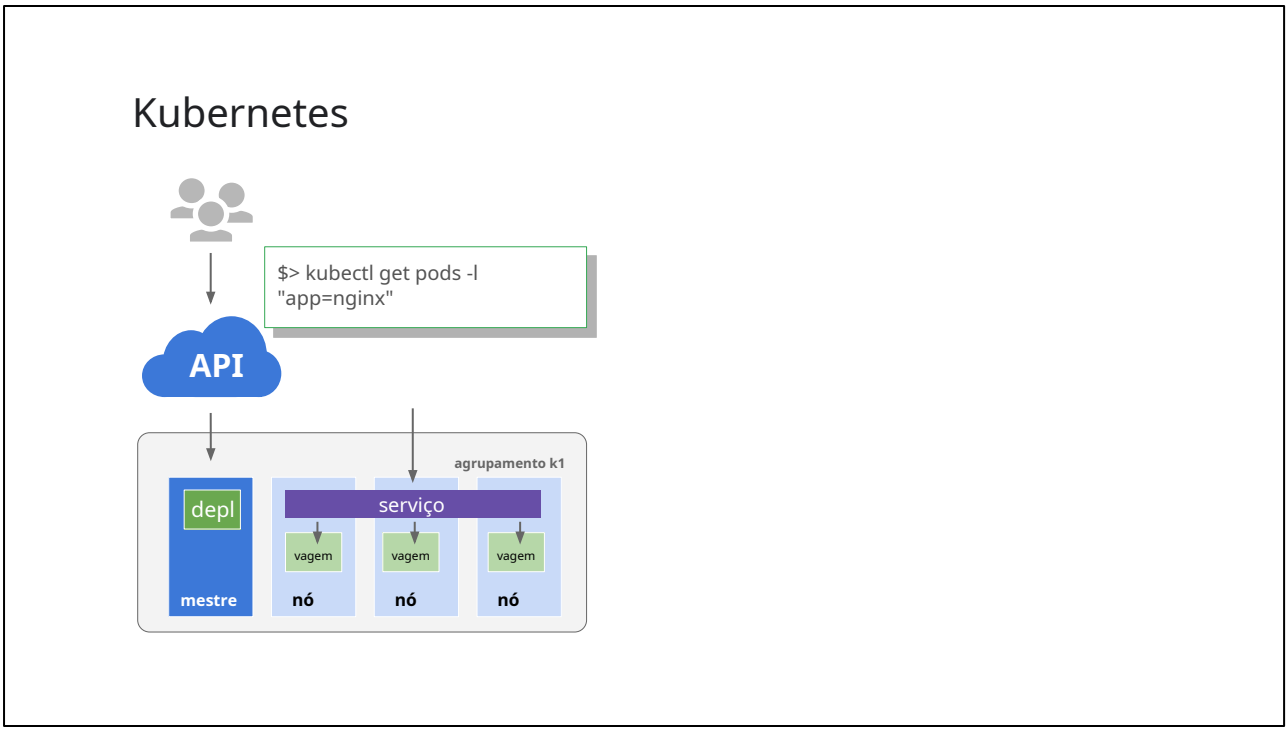
Nesse caso, três pods são criados em sua implantação e são colocados atrás do serviço e compartilham um IP fixo.

8:30



Você também poderia usar**escalonamento automático**com todos os tipos de parâmetros.

Veja um exemplo de como dimensionar automaticamente a implantação para entre 10 e 15 pods quando a utilização da CPU atingir 80%.



Até agora, eu mostrei a você como correr**imperativo**comandos como**expor**e**escala**. Isso funciona bem para aprender e testar o Kubernetes passo a passo.

Mas a verdadeira força do Kubernetes vem quando você trabalha em um**declarativo**maneira.

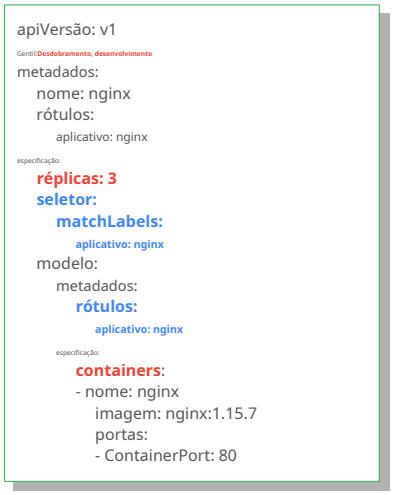
Em vez de emitir comandos, você fornece um arquivo de configuração que informa ao Kubernetes como você deseja que o estado desejado seja, e o Kubernetes descobre como fazê-lo.

Deixe-me mostrar como dimensionar sua implantação usando um arquivo de configuração de implantação existente.

Para obter o arquivo, você pode executar um comando kubectl get pods como o seguinte.

9:00

Kubernetes



E você obterá um arquivo de configuração de implantação como o seguinte.

Nesse caso, ele declara que você deseja três réplicas do seu nginx Pod.

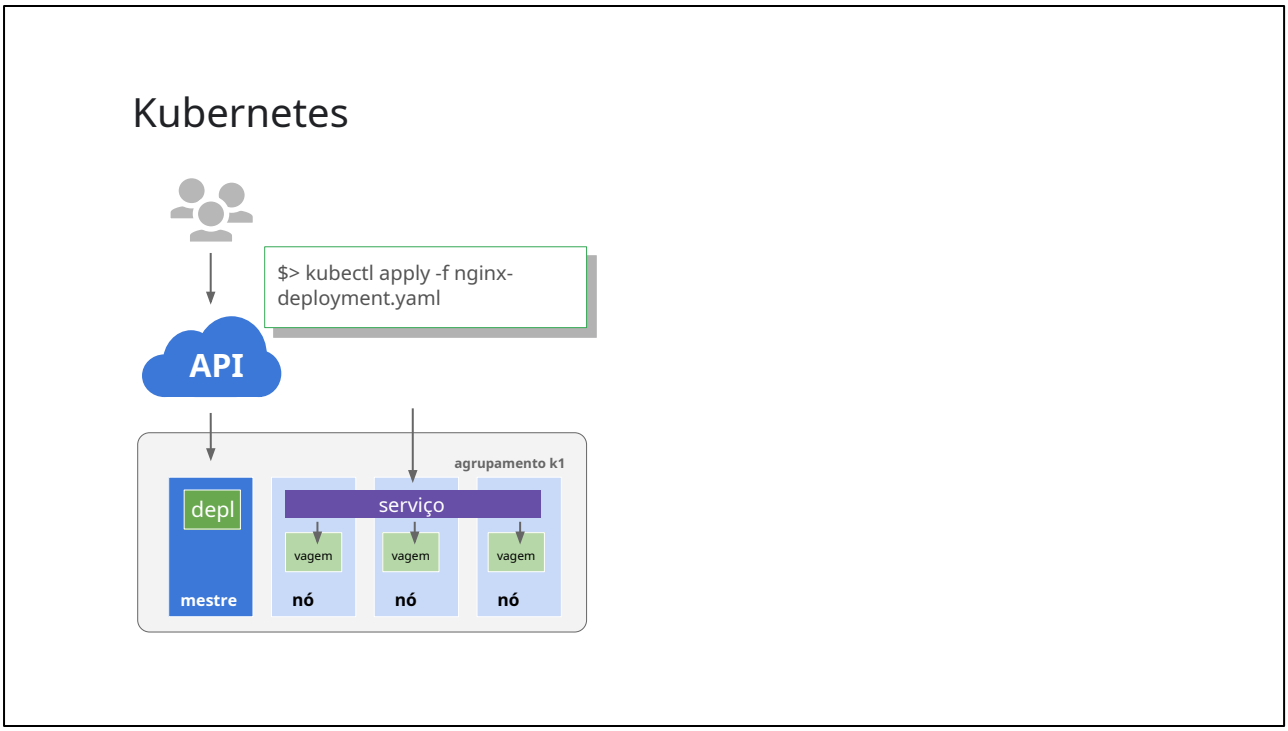
Ele define um**seletor**campo para que sua implantação saiba como agrupar pods específicos como réplicas e você adiciona um**rótulo**ao modelo de pod para que sejam selecionados.

9:15

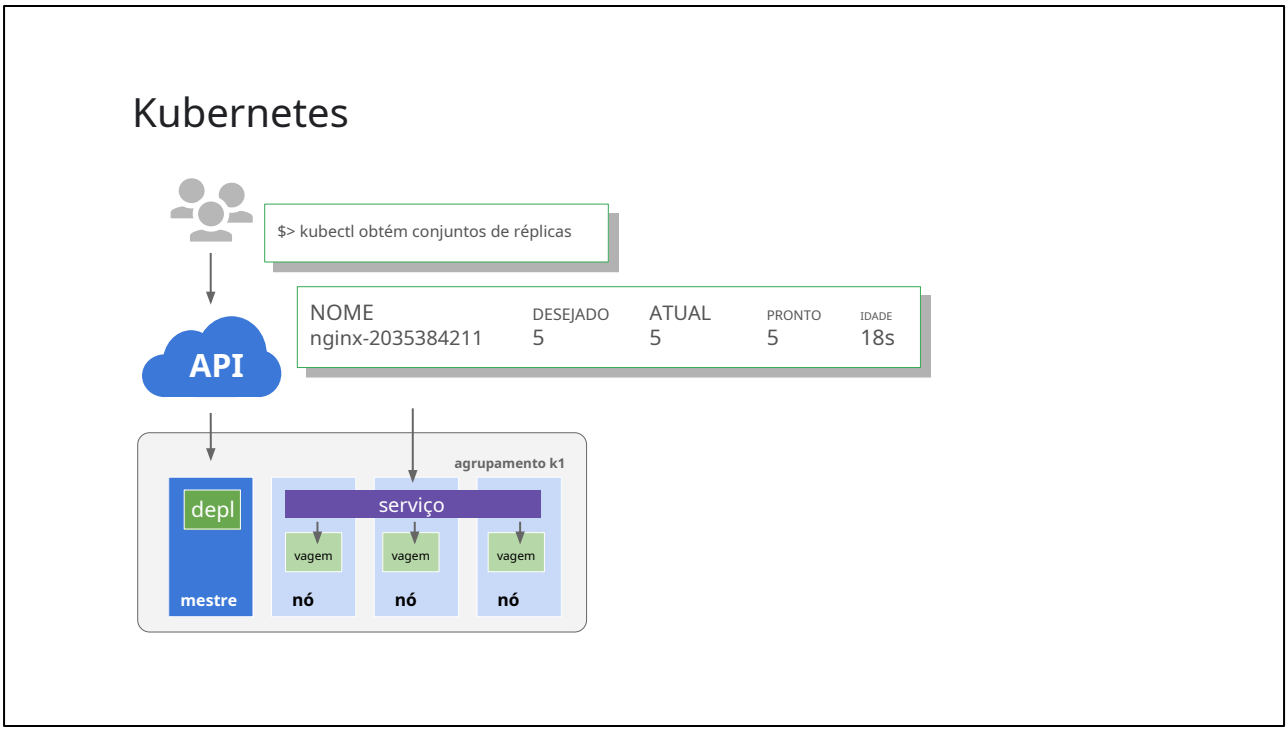
Kubernetes

```
apiVersão: v1
tipo: implantação
metadados:
  nome: nginx
  rótulos:
    aplicativo: nginx
especificação:
  réplicas: 5
  seletor:
    matchLabels:
      aplicativo: nginx
  modelo:
    metadados:
      rótulos:
        aplicativo: nginx
    especificação:
      containers:
      - nome: nginx
        imagem: nginx:1.10.0
        portas:
        - ContainerPort: 80
```

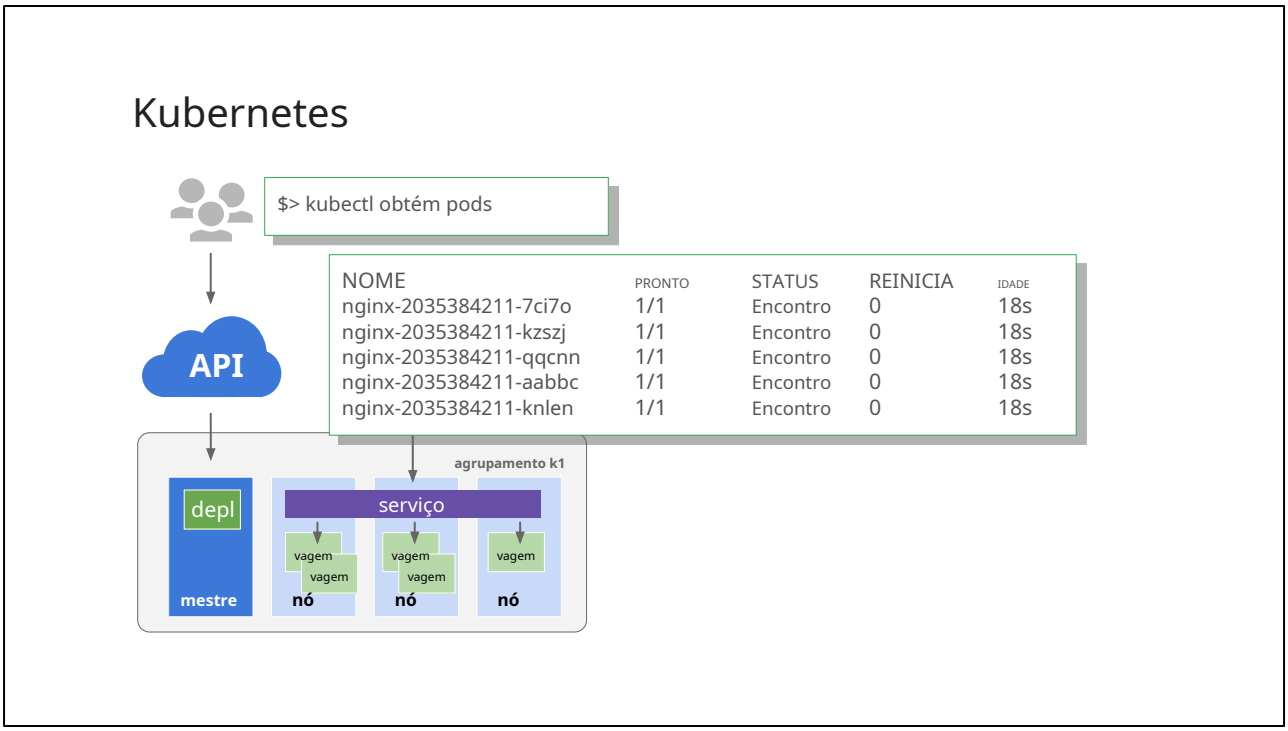
Para executar cinco réplicas em vez de três, basta atualizar o arquivo de configuração de implantação.



E execute o**kubectl aplicar**comando para usar o arquivo de configuração.

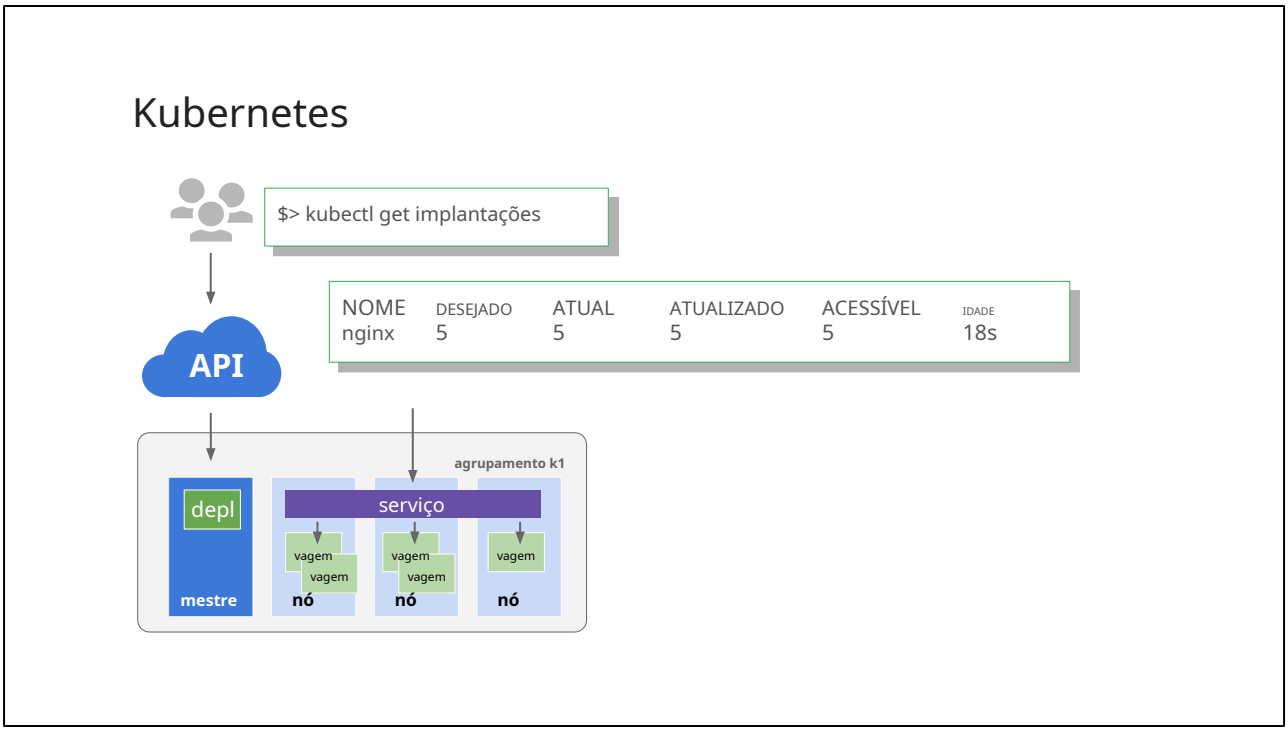


Agora olhe para suas réplicas para ver seu estado atualizado.

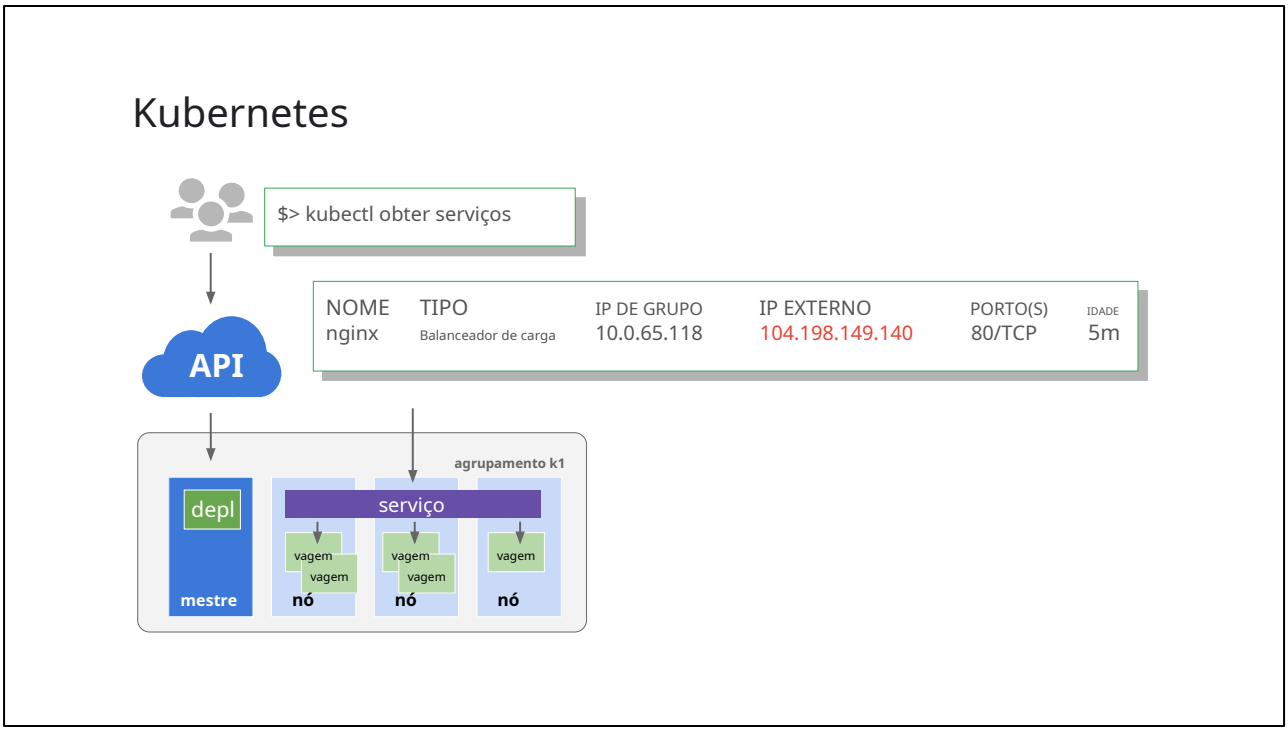


Em seguida, use o**kubectl obter pods**comando para ver os pods entrarem em linha.

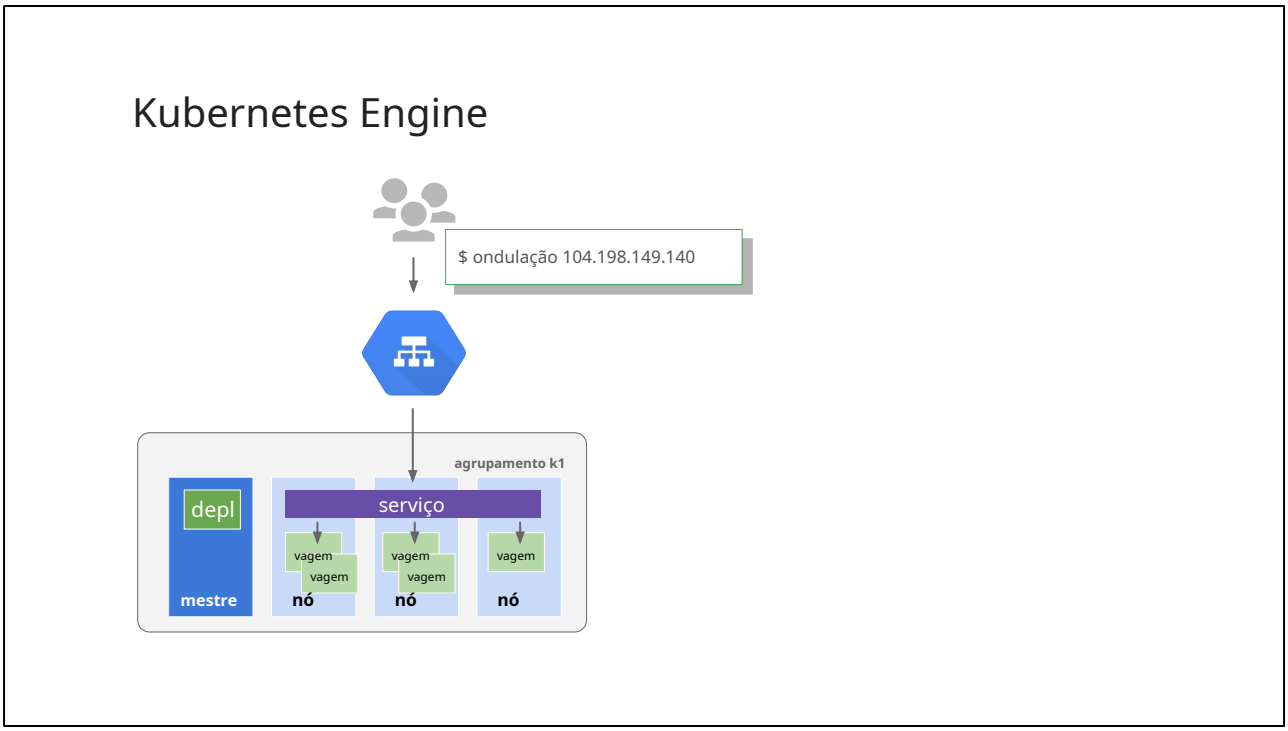
Neste caso, todos os cinco são**PRONTO**e**ENCONTRO**.



E verifique a implantação para garantir que o número adequado de réplicas esteja sendo executado usando $ kubectl**obter implantações**ou $ kubectl**descrever implantações**. Nesse caso, todas as cinco réplicas de Pod são**ACESSÍVEL**.



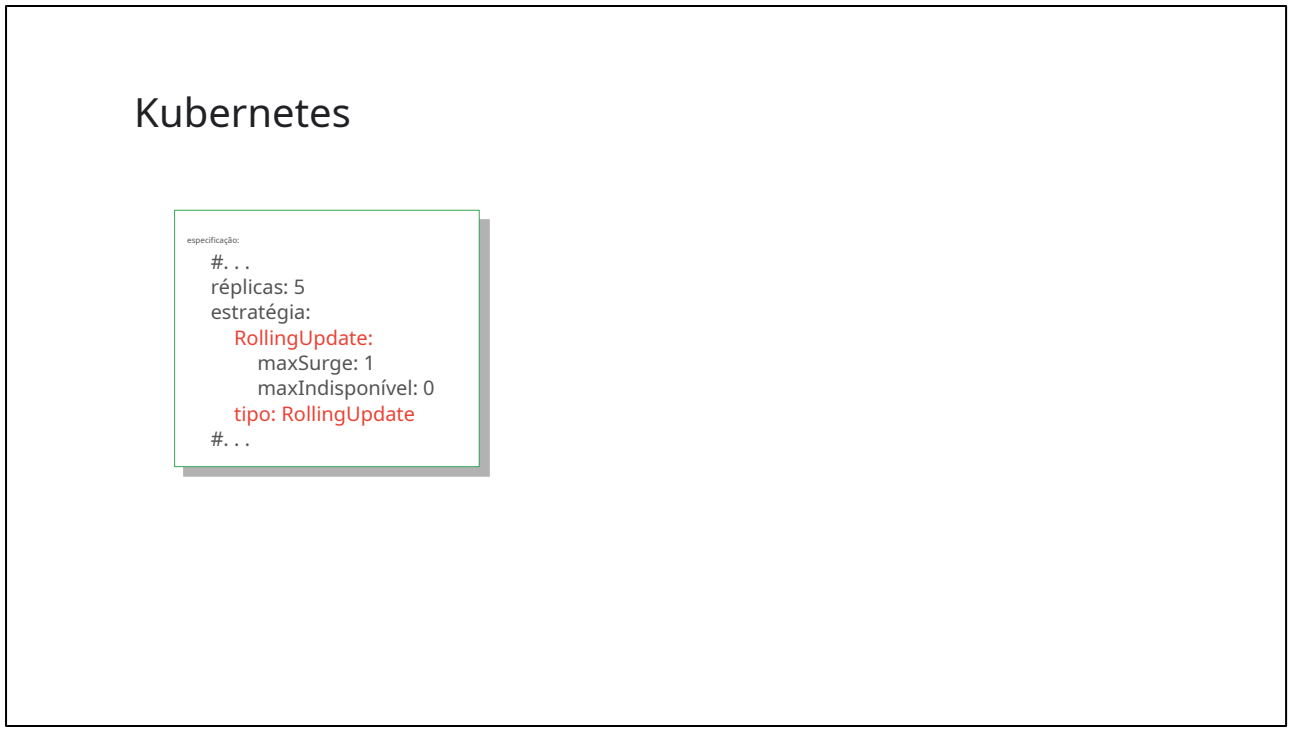
E você ainda pode atingir seu endpoint como antes de usar $ kubectl**obter serviços**para obter o IP externo do Serviço,



e acertar o IP público de um cliente.

Neste ponto, você tem cinco cópias do seu pod nginx em execução no GKE e tem um único serviço que está fazendo proxy do tráfego para todos os cinco pods. Isso permite que você compartilhe a carga e dimensione seu serviço no Kubernetes.

10:15



A última pergunta é o que acontece quando você deseja atualizar uma nova versão do seu aplicativo?

Você deseja atualizar seu contêiner para obter um novo código na frente dos usuários, mas seria arriscado implementar todas essas alterações de uma só vez.

Então você usa**lançamento do kubectl**ou altere seu arquivo de configuração de implantação e aplique a alteração usando**kubectl aplicar**.

Novos Pods serão criados de acordo com sua estratégia de atualização. Aqui está um exemplo de configuração que criará novos Pods de versão um por um e aguardará que um novo Pod esteja disponível antes de destruir um dos antigos Pods.

Nome do vídeo: T-GCPFCI-B_5_l2_LabIntroKubernetesEngine

Tipo de conteúdo: Vídeo - Apresentador de palestra

Apresentador: Jim Rambo

Laboratório

Começando com Kubernetes Engine

Jim Rambo

Há muitos recursos no Kubernetes e no GKE que ainda nem abordamos, como: configurar verificações de integridade, definir afinidade de sessão, gerenciar diferentes estratégias de lançamento e implantar pods em regiões para alta disponibilidade. Mas, por enquanto, isso é o suficiente.

Neste módulo, você aprendeu a:

- Crie e execute aplicativos em contêiner
- Orquestre e dimensione-os em um cluster
- E implemente-os usando lançamentos.

Agora você verá como fazer isso em uma demonstração e praticará em um exercício de laboratório.

11:00

Nome do vídeo:
T-GCPFCI-B_M5_L4_GettingStartedWithKubernetesEngine

Tipo de conteúdo: Vídeo - Apresentador de palestra

Apresentador: Brian Rice

Nota para o editor de vídeo: este vídeo está linkado no mapa do curso e já foi criado. Nada a editar aqui para a V2 do módulo