Engineer

Kubernetes

ett system som håller en flotta containrar igång

Vad det är

Kubernetes, ofta skrivet K8s, är programvara med öppen källkod som kör containeriserade applikationer på en grupp maskiner, ett kluster, och håller dem i linje med en beskrivning som du ger den: vilken avbildning varje applikation kör, hur många kopior som finns, hur mycket minne och processor varje kopia får använda och hur andra program når den. Projektets egen dokumentation kallar den en portabel och utbyggbar plattform med öppen källkod för att hantera containeriserade arbetslaster och tjänster. Namnet är grekiska för rorsman eller lots, K8s räknar de åtta bokstäverna mellan K och s, och Google släppte projektet som öppen källkod 2014.

Ett kluster har två halvor. Styrplanet (control plane) håller beskrivningen: API-servern är det enda HTTP-gränssnitt som varje ändring passerar, etcd lagrar det som den får veta, schemaläggaren tilldelar varje ny pod en lämplig nod, och controller manager kör de styrslingor som agerar utifrån beskrivningen. På varje arbetsnod ser kubelet till att de poddar som har tilldelats noden körs, `kube-proxy` underhåller nätverksreglerna bakom en Service, och en containerkörmiljö som containerd eller CRI-O startar containrarna. Docker Engine slutade vara en inbyggd körmiljö i Kubernetes 1.24, och avbildningar som har byggts med Docker körs ändå oförändrade, eftersom de är vanliga OCI-avbildningar.

Beskrivningen är en uppsättning objekt, oftast skrivna som YAML. En Pod är den minsta enhet som Kubernetes schemalägger: en eller flera containrar som delar nätverksadress och lagring på en maskin, i dagligt tal en pod, flera poddar. En Deployment anger hur många likadana poddar som ska finnas och byter ut dem steg för steg när deras mall ändras. En Service ger en föränderlig uppsättning poddar ett stabilt namn och en stabil adress och väljer sina medlemmar efter etikett. En liten webbapplikation ser ut så här:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: web
spec:
  replicas: 3
  selector:
    matchLabels:
      app: web
  template:
    metadata:
      labels:
        app: web
    spec:
      containers:
        - name: web
          image: registry.example.com/web:1.4.2
          resources:
            requests:
              cpu: 250m
              memory: 256Mi
            limits:
              memory: 256Mi
          readinessProbe:
            httpGet:
              path: /healthz
              port: 8080
```

Kubernetes vägs oftast mot tre mindre alternativ. Docker Compose beskriver containrar på en maskin, och när maskinen försvinner gör de det också. En tjänstehanterare som systemd kan starta om en container som har kraschat och starta den vid uppstart, vilket täcker väldigt många behov med betydligt mindre maskineri, men den har inget begrepp om att flytta arbete till en annan maskin. Hostade plattformar som Google Cloud Run eller Azure Container Apps kör containrar utan att lämna över ett styrplan som du måste driva, i utbyte mot färre inställningar. Dokumentationen säger också rakt ut vad Kubernetes inte är: en traditionell, allomfattande plattform som tjänst. Den bygger inte din applikation, och den levereras inte med några egna databaser eller meddelandeköer.

Använd begreppet i ett beslut

Använd ”Vad är Kubernetes?” i ett konkret beslut. Vi kan knyta begreppet till systemet ni bygger eller köper. Paketerad för att bete sig identiskt på en laptop, på en testserver och i produktion, och för att skala upp när en hektisk vecka kommer.

Se våra tjänster Hör av er →

Granska antagandena bakom ordet

Använd förklaringen och det länkade arbetet för att formulera frågor till projektet. Vi kan undersöka antagandena tillsammans innan ett verktyg väljs. Drev 20 Docker-appar på Google Cloud och AWS — 20 containerbaserade applikationer, driftsatta och övervakade

Drev 20 Docker-appar på Google Cloud och AWS Hör av er →

Varför det spelar roll

Den bärande idén är att klustret hela tiden arbetar mot en skriven beskrivning, i stället för att köra en lista med kommandon en gång. Att tillämpa filen ovan startar ingenting direkt; den registrerar en önskan. Controllers är styrslingor som bevakar klustret och agerar för att föra dess nuvarande tillstånd närmare det önskade. Om en maskin dör och tre kopior blir två ser controllern för din Deployment glappet, en ny pod skapas, och schemaläggaren placerar den på en maskin med utrymme. Eftersom beskrivningen är en uppsättning filer kan den ligga i ett repo och gå igenom samma granskning som koden den kör.

Releaser följer med plattformen. En Deployments förvalda strategi är en rullande uppdatering: nya poddar startar, och gamla stoppas i takt med att de nya blir redo, inom gränser som förvalt är 25 % extra och 25 % saknade vid varje tidpunkt. Varje ändring av poddmallen sparas som en revision, tio som förval, och `kubectl rollout undo` går tillbaka till den föregående. Skriptet som en gång tömde en server, bytte en binär och tog tillbaka den blir inbyggt beteende.

Schemaläggaren ändrar också ekonomin för en maskinpark. Den placerar varje pod efter den processor och det minne som podden begär, så att många små tjänster delar en uppsättning maskiner i stället för att var och en får en server dimensionerad för sin sämsta dag.

Priset är verkligt. Ett kluster är ett distribuerat system i sin egen rätt, med versioner att uppgradera enligt schema, ett nätverkslager, ett lagringslager, certifikat och åtkomstkontroll, och någon måste förstå allt det när det beter sig illa. Hanterade tjänster som Google Kubernetes Engine, Amazon EKS och Azure Kubernetes Service driver styrplanet åt dig, men varje manifest, och oftast arbetsnoderna, förblir ditt ansvar. För en handfull tjänster på en eller två maskiner gör en vanlig tjänstehanterare ofta jobbet med färre rörliga delar.

Så ser det ut i praktiken

Företagets sida om teknikstacken säger regeln rakt ut: Docker och Kubernetes för de arbetslaster som förtjänar dem, och vanliga systemd-enheter för dem som inte gör det. Kubernetes väljs per arbetslast, efter jobbets storlek och form, och

inte som förval.

Arbetslaster som förtjänade det finns i ett uppdrag. Tjugo Docker-applikationer driftsattes och underhölls i Google Cloud och AWS, däribland några R-baserade analysappar, med enhetlig konfiguration och övervakning så att ingen applikation blev ett specialfall. Felsökningen gjordes med Podman, och orkestreringen felsöktes med Kubernetes. Sidan redovisar att problem diagnostiserades snabbare och att driftsättningarna förblev stabila och reproducerbara.

Den andra halvan av regeln syns på företagets egen plattform. En containermiljö byggd på Podman och Quadlet kör varje container som en tjänst under operativsystemets egen tjänstehanterare, beskriven i samma deklarativa form som allt annat på värden. Containrarna använder värdnätverk och publicerar inga portar, så brandväggsreglerna som härdningsrollen installerar behåller kontrollen. En release görs av en användare utan privilegier, som kör ett och samma fasta skript; skriptet bygger avbildningen och startar om tjänsten. På en enda värd ger det varje container en skriven beskrivning och den tjänstehanterare som värden redan litar på, utan någon andra processhanterare. Det som lämnas utanför är placering över flera maskiner och att ersätta en död nod, och det är där Kubernetes börjar betala för sin vikt.

Var det går fel

De flesta problem med Kubernetes kommer ur glappet mellan vad ett manifest begär och vad klustret rapporterar tillbaka.

En utrullning står halvfärdig medan pipelinen rapporterar framgång. När nya poddar aldrig blir redo, på grund av fel avbildningstagg, en krasch vid start, en readiness-prob som fallerar eller en kvot som tar slut, slutar din Deployment att göra framsteg. Efter tidsgränsen för framsteg, som förvalt är 600 sekunder, registrerar den orsaken `ProgressDeadlineExceeded`, och dokumentationen säger att Kubernetes inte vidtar någon annan åtgärd: de gamla poddar som fortfarande körs fortsätter att svara, och ingenting rullas tillbaka av sig självt. `kubectl rollout status` avslutas med en nollskild kod när tidsgränsen har passerats, så ett driftsättningssteg kan vänta på det och ångra releasen:

```
kubectl apply -f web.yaml
kubectl rollout status deployment/web --timeout=10m \
  || kubectl rollout undo deployment/web
```

En container startas om med `OOMKilled`, eller går långsamt på en nod med ledig processorkraft. Systemkärnan upprätthåller gränserna, och de två resurserna beter sig olika. En container över sin minnesgräns kan dödas och startas om; en som ligger vid sin processorgräns stryps, så den går långsammare utan att dö, vilket ser ut som ett problem i applikationen. Schemaläggaren placerar poddar efter vad de begär, inte efter vad de använder, så låga begäranden tränger ihop en nod och höga lämnar maskiner sysslolösa. En gräns som anges utan en begäran blir begäran. Sätt minnet utifrån uppmätt användning, och behandla processorgränser som ett medvetet beslut.

En applikation som startar långsamt dödas innan den har startat färdigt. En liveness-prob som fallerar får kubelet att döda och starta om containern, så en applikation som behöver en minut för att läsa in sig, och som kontrolleras efter tio sekunder, startar om i all evighet. En startup-prob håller de andra kontrollerna borta tills applikationen är uppe. En readiness-prob tar ut en pod ur dess Service utan att starta om den, vilket passar en pod som är upptagen eller värmer upp en cache. En liveness-prob som anropar databasen startar om varje replik på en gång när databasen är långsam, och gör ett långsamt beroende till ett driftstopp.

En pod som startats för hand försvinner, och en rättning gjord för hand går förlorad. En pod som skapats på egen hand ligger kvar på sin nod tills den avslutas, raderas eller vräks, eller tills noden går sönder, och ingen controller skapar den på nytt; dokumentationen råder till en Deployment eller ett Job även för en enda kopia. En ändring gjord med `kubectl edit` lever bara i klustret, och nästa tillämpning från repot lägger tillbaka det gamla värdet, så en nödrättning hör hemma i filerna så snart nödläget är över.

Läs mer

- Kubernetes overview — projektets egen redogörelse för vad Kubernetes är, vad det erbjuder och vad det lämnar utanför.
- Pods — den minsta enhet som Kubernetes schemalägger, dess livscykel och varför poddar sällan skapas för hand.
- Deployments — rullande uppdateringar, återställningar, revisionshistorik och tidsgränsen för framsteg.
- Service — stabila namn och adresser framför en föränderlig uppsättning poddar.
- Resource Management for Pods and Containers — begäranden, gränser och hur kärnan upprätthåller var och en.
- Configure Liveness, Readiness and Startup Probes — de tre proberna och hur en container som startar långsamt skyddas.

Ordlista

Välj utifrån era faktiska förutsättningar

Ta med driftsättningsvägen, driftsriskerna och dem som har jour. Vi kan avgränsa en förändring med kontroller, en väg tillbaka och tydligt ansvar.

Se vad det kostar Hör av er →

Gör begreppen praktiska

Fråga hur ”Vad är Kubernetes?” gäller i er organisation. Ett konkret exempel från ert nuvarande system räcker för att börja samtalet.

Se våra tjänster

welcome@engineer.company +45 50 10 61 84 Köpenhamn, Danmark

Ta med dig den här sidan

Visitkort

Omsorgsfullt skapad med kärlek av D.

CVR 45969703 · Tagensvej 105C, 3. th, 2200 København N

Upphovsrätt © 2025 – nu · Engineer ApS · Alla rättigheter förbehållna.

Utskrivet från https://engineer.company/sv/glossary/kubernetes/