بخش دوم

کسرا نوربخش 810100230

هومان مشیری امین 810100210

# مقدمه

در این بخش از تمرین به کار با k8s پرداختیم.

**خب دیگه دست به کیبورد شیم!**

در ابتدا با استفاده از دو دستور curl، kind و kubectl را بر روی WSL مان نصب کردیم. سپس یک فایل yml ساده با توضیحات داده شده نوشتیم(اسم فایل کانفیگ kind-ha-cluster.yaml می باشد):

kind: Cluster

apiVersion: kind.x-k8s.io/v1alpha4

nodes:

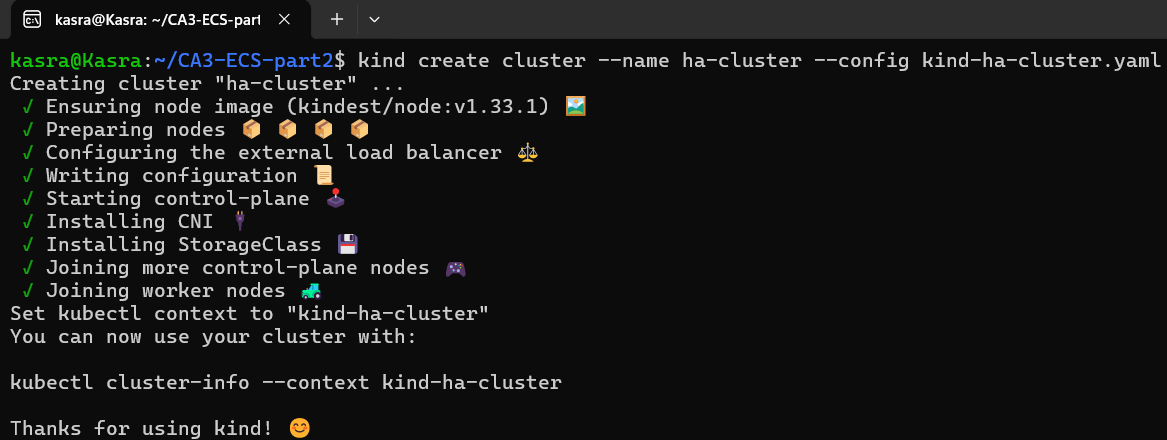
- role: control-plane

- role: control-plane

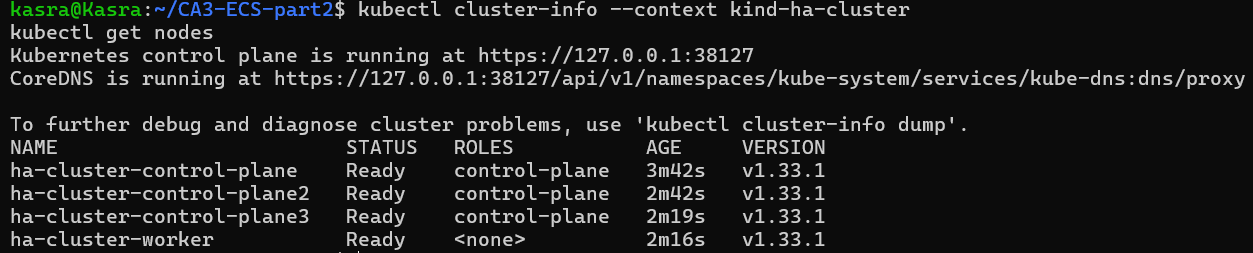
- role: control-plane

- role: worker

سپس با استفاده از دستور زیر، کلاستر را ساختیم و deploy کردیم:



اطمینان حاصل کردیم که ساخته شده باشند:



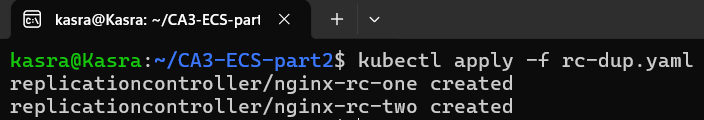
می دانیم که در هر سیستم توزیع شده ای ما نیاز به اجماع رسیدن و الگوریتم های Quorumداریم و برای این که بتوانیم مابین master ها در شرایط رخداد یک حادثه، ارور یا هر جای دیگری به نتیجه گیری برسیم، تعداد باید فرد باشد. Best practice برای تعداد نود های مختلف در کلاستر های بسیار بزرگ: مستر نود ها: 3 یا 5. به دلیل این که با داشتن 3 نود مستر تقریباً تمامی نیاز های سازمان تامین می شود اما اگر دسترس پذیری بیشتر بخواهیم به سراغ 5 می رویم. اگر بیش از 5 داشته باشیم موجب تاخیر و پیچیدگی می شود. ورکر نود ها: بسته به workload ای که داریم مقیاس می شوند و می توانند بین 10 تا بیش از 1000 نود هم باشند. ضمناً برای نگه داری وضعیت کلاستر از 3 الی 5 etcd هم استفاده می شود.

**آپدیت ما نباید مشتری رو اذیت کنه!**

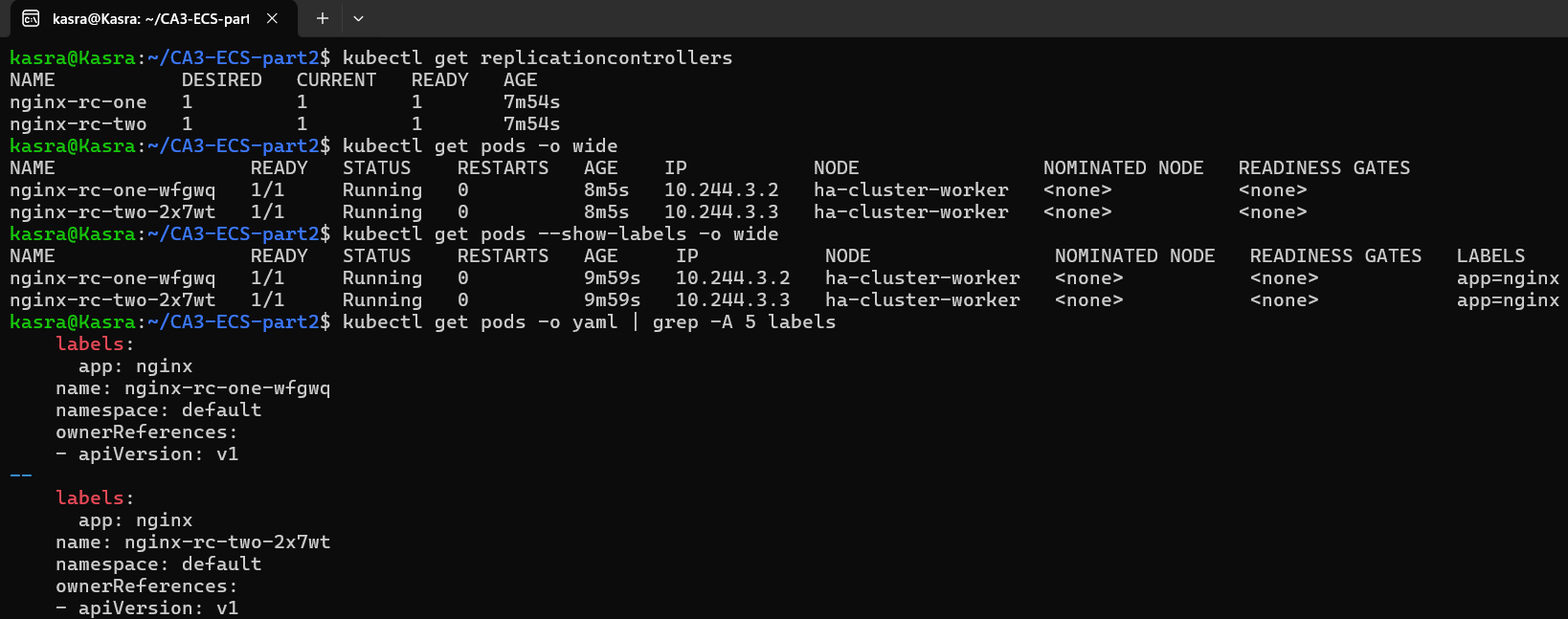
در واقع هر دو کنترل کننده هایی هستند که اطمینان حاصل می کنند که تعداد مشخصی pod در هر لحظه در حال اجراست. ReplicationController مدل قدیمی تر است که ساده تر هم می باشد و از یک single label selector برای شناسایی pod ها استفاده می کند.(بنابراین نمی تواند به صورت پیچیده میان آن ها انتخاب کند) اما ReplicaSet ها پیشرفته تر اند و از set-based selector ها استفاده می کنند که عموماً توسط کنترلر های سطح بالاتر مانند deployment ها مدیریت می شوند و در انتخاب pod ها انعطاف پذیر تر اند. ما در deployment های جدید نباید از ReplicationController استفاده کنیم.

با توجه به توضیحات داده شده، هر ReplicationController تلاش می کند که به صورت مستقل، 1 پاد را maintain کند که بتواند با selector اش آن را انتخاب کند. چون تمامی تنظیماتشان یکسان می باشد، هر دو تلاش می کنند که یک pod مشخص را انتخاب کنند، پس اولین کنترلر آن را می سازد و دومین کنترلر هم pod ای را می سازد اما مساله در این است که به صورت اتوماتیک هنگام ساختن pod ها یک pod-template-hash هم پشت صحنه اضافه می شود که از هم قابل تمییز دادن هستند.(البته اول من از hash پشت صحنه خبر نداشتم و انتظار داشتم 1 pod ساخته بشه اما 2 تا شد)

ابتدا manifest داده شده را ذخیره کردیم و با استفاده از دستور زیر آن را apply کردیم:



حال می توانیم خروجی را ببینیم: (باید دقت کرد که گرچه label ها یکسان اند اما در انتها می بینیم که hash به آن ها اضافه شده است و name ها متفاوت است)



اگر از ReplicationController داریم استفاده می کنیم چاره این است که در manifest، در قسمت image نام را از 20 به 21 تغییر بدهیم یعنی: nginx:1.21 و دوباره دستور apply رو بزنیم و به صورت دستی pod های قدیمی را پاک بکنیم که این روش بسیار بدی می باشد چون همراه با down time می باشد و به صورت دستی باید اعمال شود. اگر به سراغ ReplicaSet ها برویم، می توانیم باز دستی خط مربوط به image را تغییر دهیم و مقیاس به پایین و بالا به ترتیب داشته باشیم که down time کاهش می یابد اما باز هم دستی داریم عمل می کنیم که بد است. بهترین روش استفاده از deployment می باشد که پیش تر گفتیم ReplicaSet را هم کنترل می کند به این صورت که manifest دپلویمنت را ادیت می کنیم و apply می زنیم. در این حالت k8s به صورت اتومات rolling update را اجرا می کند به این معنا که به آرامی پاد های جدید را مقیاس می کند و و هر وقت ready شدند، پاد های قدیمی را به 0 مقیاس می کند. با این روش هم down time نداریم، هم از منابع به خوبی استفاده کرده ایم و هم rollback می توانیم بکنیم. دستور apply: kubectl apply -f deployment.yaml می باشد و فایل yaml شامل:

spec:

template:

spec:

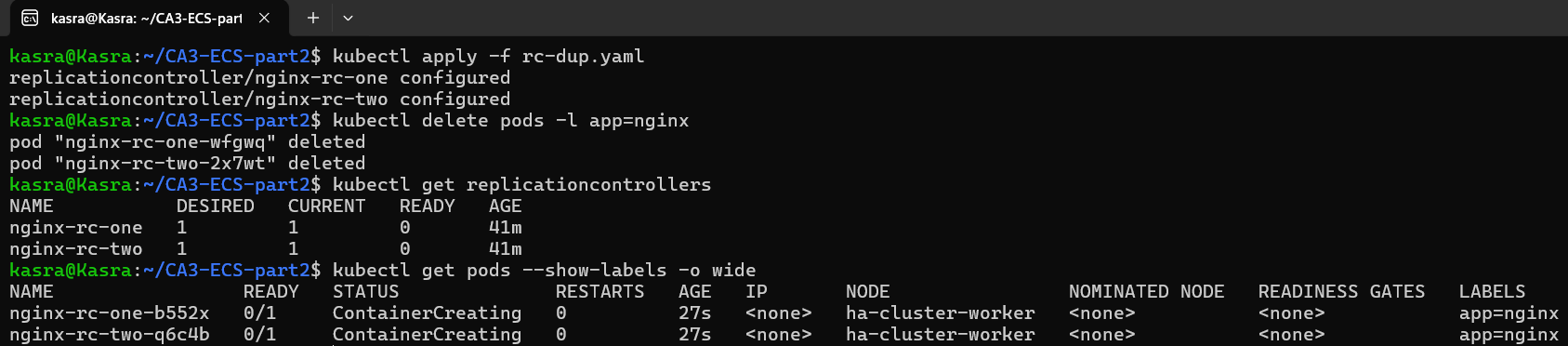
containers:

- name: nginx

image: nginx:1.21 # Update version

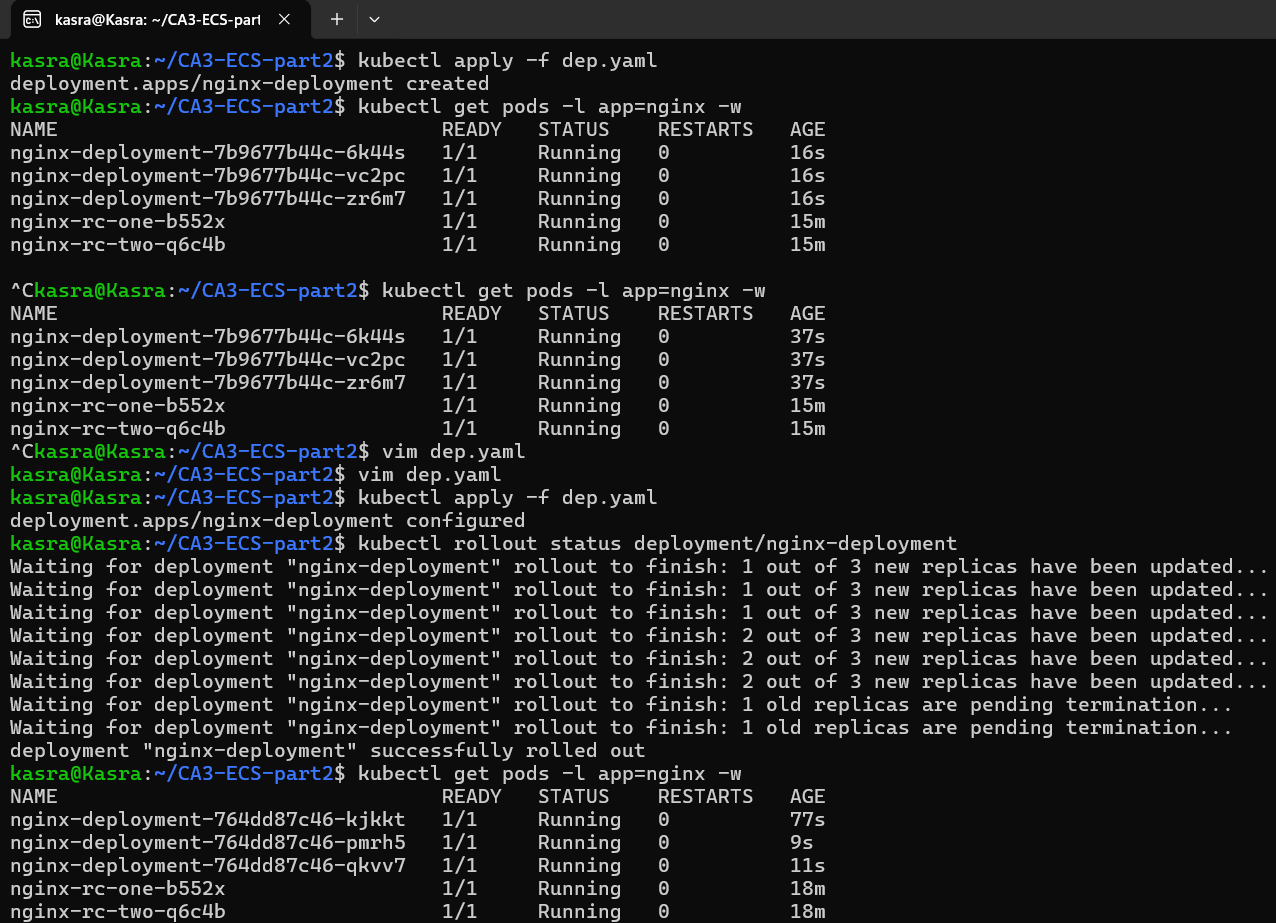
خواهد بود.

سناریو آپدیت به همراه ReplicationController به صورت زیر است که دستی image را تغییر می دهیم:

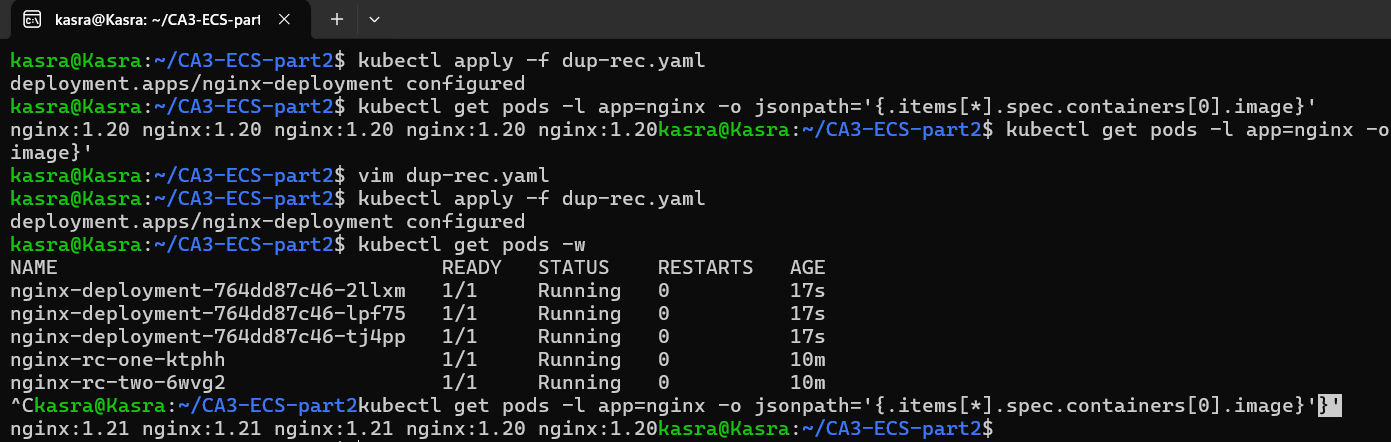


با استفاده از Deployment ما می توانیم Declarative و Rolling آپدیت داشته باشیم و همچنین به ورژن قبلی rollback کنیم اما هیچ کدام از این ها را در ReplicationController نداریم. برای انتخاب کردن یک pod هم به مانند ReplicationController می باشد.

برای سناریو آپدیت با استفاده از Deployment یک فایل manifest با نام dep.yaml ساختیم که ابتدا نسخه nginx آن 20 بود. سپس آن را دستی 21 کردیم و منتظر rollout شدیم:



Recreate در واقع قبل از ایجاد کردن pod های جدید، قدیمی ها را می کشد اما rolling update ها به صورت ترتیبی جایگزین می کند قدیمی ها را با جدید ها که در سناریو بالا این را دیدیم. حال به سراغ دیدن نحوه عمل کرد recreate می رویم، برای این کار یک dup-rec.yaml ایجاد کردیم و جلو رفتیم که مشاهده می شود 21 شدند:



**سوالات تشریحی**

1. PV مخفف Persistent Volume هست که یک storage یا همون حافظه در cluster که توسط یک ادمین و یا به صورت dynamic، فراهم میشود. PVC مخفف Persistent Volume Claim هست که در واقع درخواستی هست که کاربر برای حافظه میدهد. (مثل اندازه و مد دسترسی و ...) کاربرد StorageClass در abstraction ای هست که برای بک اند حافظه فراهم می کند و همچنین اتومات کردن فرآیند ایجاد PV بر اساس درخواست های PVC. در واقع مشخص می کنه که حافظه چطور به صورت پویا اختصاص داده میشه.

2. CNI مخفف Container Network Interface هست که در واقع یک تعریف برای کانفیگ کردن اینترفیس های شبکه در Linux container ها می باشد. CNI ها شبکه رو توی k8s فراهم می کنند. Cilium از eBPF استفاده می کنه و پالیسی های پیشرفته شبکه رو هم حتی ساپورت می کند(حتی امنیت) اما Calico از eBPF یا iptables استفاده می کنه و ساده تر هست و مناسب مقیاس پذیری و policy enforcement هست. به طور خلاصه Cilium پیشرفته تر و پر فیچر تره، Calico بیشتر استفاده شده و آداپته شدست.

3.

4. KEDA مخفف Kubernetes-based Event Driven Autoscaler هست. پاد ها رو بر اساس رخداد ها(events)، مثل طول صف و تعداد درخواست های http و یا متریک های خارجی scale می کند. با HPA کار می کند اما اون رو به یک سیستم event-driven گسترش می دهد(مثل Kafka). کاربردش توی Serverless workloads ها، معماری ها و سیستم های event-driven و کم کردن هزینه scale کردن می باشد.

# لینک های استفاده شده

# 1. کانفیگ kind: [https://kind.sigs.k8s.io/docs/user/configuration/](https://kind.sigs.k8s.io/docs/user/configuration/%20)

2. پاسخ به سوالات پرسیده شده: **تمامی لینک های قرار داده شده در صورت پروژه**