In the name of God

Essential Computer Skills

CA 3

Golboo Rashidi - 810100148

Erfan Ahmadi -0810100078

Part 2 - K8s

---

1. Warm-up!

• Kind Installation

```
thegreatgolboo@golboo:~/Desktop/ECS/CA3/P2$ curl -Lo ./kind https://github.com/kubernetes-sigs/kind/releases/latest/download/kind-linux-amd64
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
  0     0    0     0    0     0      0      0 --:--:-- --:--:-- --:--:--     0
  0     0    0     0    0     0      0      0 --:--:-- --:--:-- --:--:--     0
100 10.5M  100 10.5M    0     0  1286k      0  0:00:08  0:00:08 --:--:-- 1451k
thegreatgolboo@golboo:~/Desktop/ECS/CA3/P2$ chmod +x ./kind
thegreatgolboo@golboo:~/Desktop/ECS/CA3/P2$ sudo mv ./kind /usr/local/bin/kind
thegreatgolboo@golboo:~/Desktop/ECS/CA3/P2$ kind --version
kind version 0.29.0
thegreatgolboo@golboo:~/Desktop/ECS/CA3/P2$
```

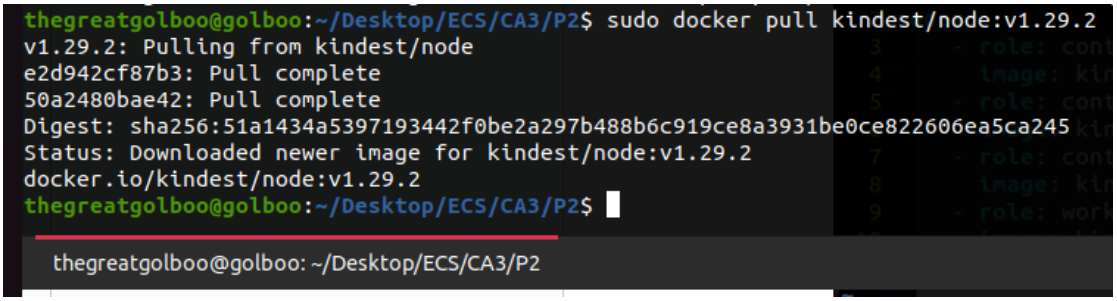
thegreatgolboo@golboo: ~/Desktop/ECS/CA3/P2

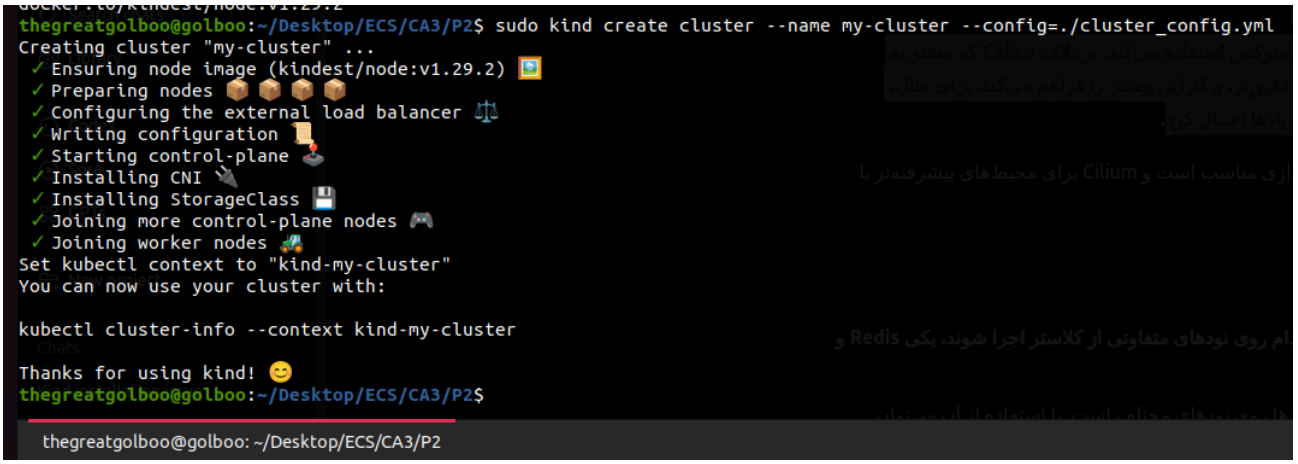
• Kind config

We created cluster_config.yml with the following config:

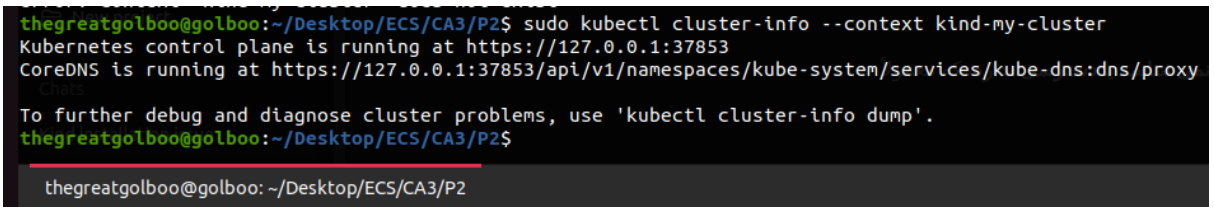
```yaml
kind: Cluster
apiVersion: kind.x-k8s.io/v1alpha4
nodes:
  - role: control-plane
    image: kindest/node:v1.29.2
  - role: control-plane
    image: kindest/node:v1.29.2
  - role: control-plane
    image: kindest/node:v1.29.2
  - role: worker
    image: kindest/node:v1.29.2
```

Creating cluster:

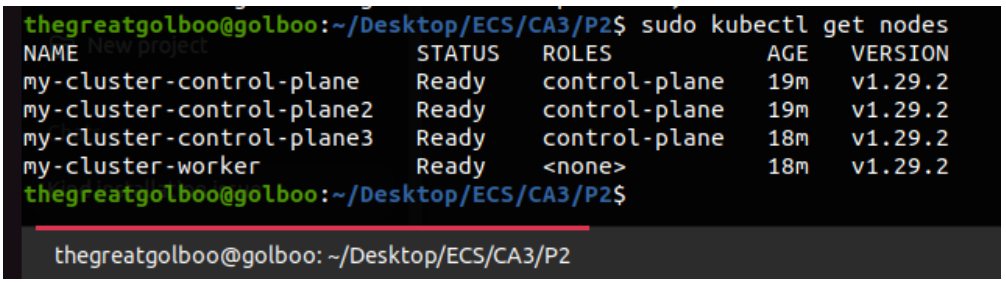




Verify cluster creation:



We can see the nodes:



- **Odd number of Master nodes**

In K8s, the master nodes(control plane nodes) are responsible for managing the cluster. They handle scheduling, monitoring cluster state, and making critical decisions.

A key component of the control plane is etcd, which is a distributed key-value store where the entire state of the cluster is stored. To maintain consistency and avoid split-brain scenarios, etcd uses a consensus algorithm like Raft. Raft requires a majority of nodes to agree on changes. For example:

- If you have 3 master nodes, a majority (2 out of 3) is needed to make a decision.
- If 1 node fails, the remaining 2 can still form a majority and continue operating.

But if we have an even number of master nodes, and one fails, we lose majority and the system can no longer make decisions, effectively making the cluster unresponsive at the control plane level.

- **Number of Master and Worker Nodes in Enterprise-Grade Clusters**

  In small or development environments, you might see: 1 master node and, 1–2 worker nodes

  But in enterprise or production-level systems, the architecture is more robust:

  - 3 or 5 master nodes are typically used for high availability.(3 is most common for production environments.)
  - Dozens to hundreds of worker nodes are used depending on application load. These worker nodes may be grouped by workload type (memory-intensive, CPU-intensive).

- Large organizations use multi-cluster management tools like Rancher, KubeFed, or managed Kubernetes services (GKE, EKS, AKS) to handle thousands of nodes.

For example, Google and AWS have production clusters that handle 5,000+ worker nodes in a single environment made possible through advanced automation and monitoring tools.

2. Our update should not bother the customer!

This line emphasizes a key principle in modern software deployment: “Updates should not cause downtime or user disruption.”

To ensure this in Kubernetes, we rely on mechanisms like the Replication Controller (RC) and ReplicaSet (RS) to manage application availability during updates.

- **Difference Between Replication Controller and ReplicaSet:**
  - Replication Controller (RC): Replication Controller was the original mechanism in Kubernetes to ensure that a specified number of Pod replicas are always running. For example, if you declare 3 replicas, the RC will make sure that 3 Pods are running at all times, recreating them if they crash or get deleted. However, RCs have limited label selectors, making them less flexible and harder to manage in complex scenarios.
  - ReplicaSet (RS): ReplicaSet is the newer version of the Replication Controller. It does the same core job, ensuring the desired number of Pods are running, but with advanced label selectors for better control. Most importantly, ReplicaSets are the foundation of Deployments in Kubernetes. When you create a Deployment, it

automatically creates and manages a ReplicaSet behind the scenes to handle scaling and updates.

- **How many Pods will exist in the cluster?**

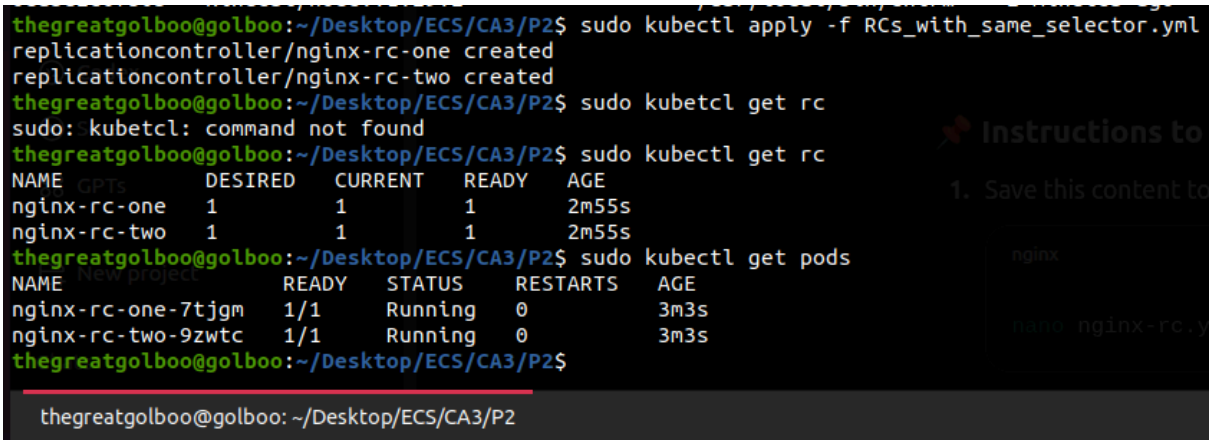
When we apply two RCs with the same selector but different names, K8s doesn't consider them duplicates. It assumes we're intentionally creating two separate controllers.

Each RC manages Pods independently. So even though the second RC sees a Pod already running with the same labels, it doesn't take over because that Pod wasn't created by that RC.

Therefore, the second RC creates another Pod to satisfy its own replicas: 1 rule. This leads to two identical Pods (in spec), but each controlled by a different RC.

So, we'll end up with 2 Pods in the cluster. one from each RC.

To verify our answer we put the given manifest in RCs_with_same_selectro.yml file and then we applied it:



We can see we have 2 pods with the same labels but different names. More detail:

```
thegreatgolboo@golboo:~/Desktop/ECS/CA3/P2$ sudo kubectl get pods -o wide --show-labels
NAME                READY   STATUS    RESTARTS   AGE     IP           NODE                NOMINATED NODE   READINESS GATES   LABELS
nginx-rc-one-7tjgm  1/1     Running   0          6m13s   10.244.3.2   my-cluster-worker   <none>           <none>            app=nginx
nginx-rc-two-9zwtc  1/1     Running   0          6m13s   10.244.3.3   my-cluster-worker   <none>           <none>            app=nginx
thegreatgolboo@golboo:~/Desktop/ECS/CA3/P2$
thegreatgolboo@golboo: ~/Desktop/ECS/CA3/P2
```

- **Develop and update!**
  - **Using Replication Controller:**

    ReplicationController does not support rolling updates natively. So if we use RC and want to upgrade an app (from nginx:1.20 to nginx:1.21), we have to manually handle the transition. What we need to do is:

    1. create a second RC with the new image version
    2. scale down the old RC
    3. scale up the new RC to simulate a rolling update manually

```
thegreatgolboo@golboo:~/Desktop/ECS/CA3/P2$ sudo kubectl apply -f nginx_deployment_with_RC_v1.yml
[sudo] password for thegreatgolboo:
deployment.apps/nginx-deploy created
thegreatgolboo@golboo:~/Desktop/ECS/CA3/P2$ sudo kubectl apply -f nginx_deployment_with_RC_v2.yml
replicationcontroller/nginx-rc-v2 created
```

```
thegreatgolboo@golboo:~/Desktop/ECS/CA3/P2$ sudo kubectl scale rc nginx-rc-v2 --replicas=3
replicationcontroller/nginx-rc-v2 scaled
thegreatgolboo@golboo:~/Desktop/ECS/CA3/P2$ sudo kubectl scale rc nginx-rc-v1 --replicas=0
replicationcontroller/nginx-rc-v1 scaled
thegreatgolboo@golboo:~/Desktop/ECS/CA3/P2$ sudo kubectl get rc
NAME           DESIRED   CURRENT   READY   AGE
nginx-rc-one   1         1         1       120m
nginx-rc-two   1         1         1       120m
nginx-rc-v1    0         0         0       4m29s
nginx-rc-v2    3         3         3       14m
thegreatgolboo@golboo:~/Desktop/ECS/CA3/P2$
thegreatgolboo@golboo: ~/Desktop/ECS/CA3/P2
```

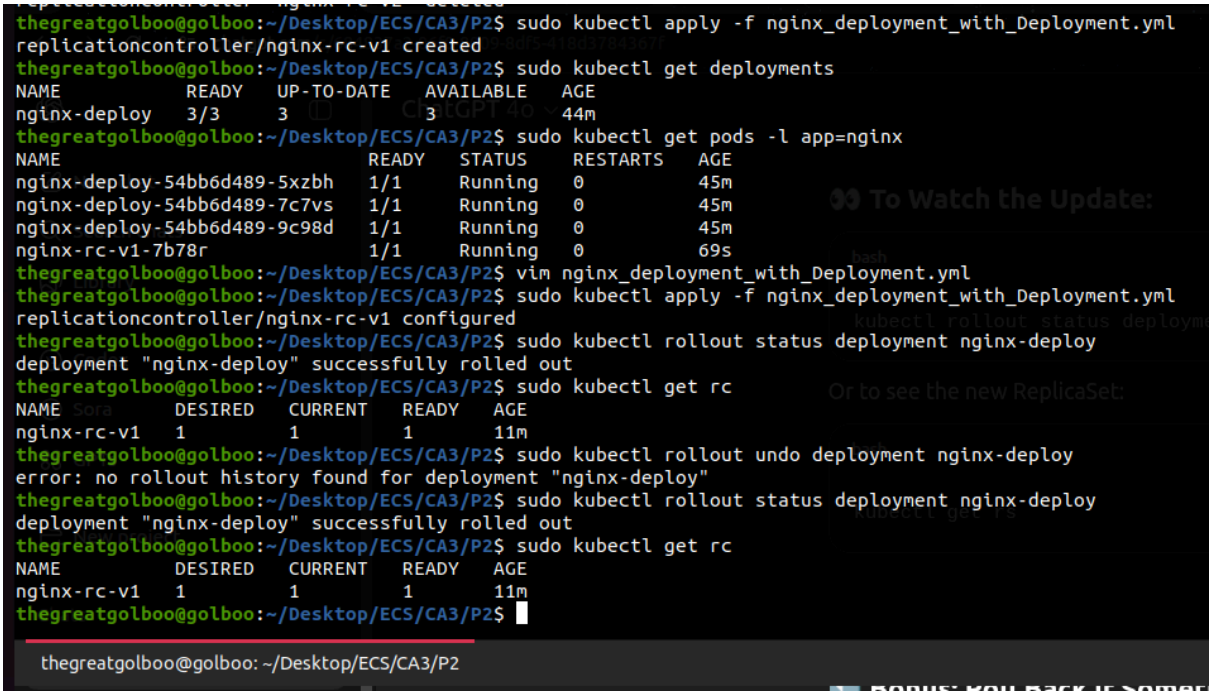
  - **Using ReplicaSet:**

    switching from ReplicationController to ReplicaSet, gives us more power (like label selectors and integration with Deployments) but the approach is the same as using ReplicaController.

  - **Using Deployment:**

using a Deployment is the best and most modern way to manage rolling updates in Kubernetes. It automates everything: scaling, updating, rollback, and availability and no manual scaling or multiple yamls needed.

What we did: We created a Deployment called nginx-deploy then applied it and checked it. Then we edit yaml file and change nginx:1.20 to nginx:1.21 and re-apply it. Here, K8s will automatically create new Pods with nginx:1.21, slowly remove the old pods (1.20), and ensure at least 1 pod stays available at all times. We watch the update using 'rollout status'. Also, if an update goes wrong, we can roll back easily.



Differences of Deployment and ReplicaSet:

A ReplicaSet is a lower-level K8s object whose main job is to make sure a certain number of identical Pods are running at all times. It uses label selectors to identify and manage the Pods it controls.

A Deployment is built on top of ReplicaSets. It provides a higher-level way to manage updates, rollouts, and rollbacks. When we create a Deployment, K8s automatically creates and manages a ReplicaSet behind the scenes. The Deployment watches over it and handles things like gradually updating the Pods when you change the image version, or rolling back to a previous version if something goes wrong.

So the key difference is that ReplicaSet only handles the number of Pods, while Deployment handles versioning, upgrades, and rollout strategies.

Rollout strategies in Deployment:

In K8s, when we update a Deployment the platform uses a rollout strategy to decide how the update should happen. There are two main rollout strategies:

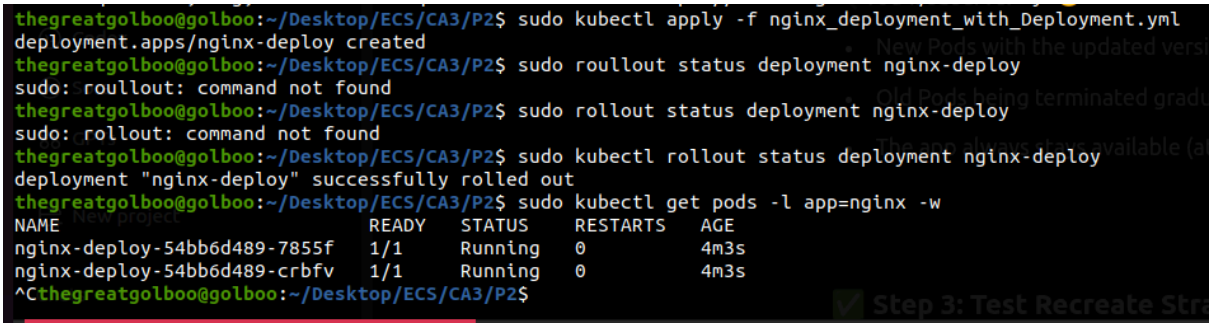
- RollingUpdate: RollingUpdate is the default and most commonly used strategy. In this method, Kubernetes updates the Pods gradually. It creates new Pods with the updated configuration while slowly terminating the old ones. This ensures that the application remains available throughout the update process, making it ideal for production environments where uptime is important. You can also control how many Pods are replaced at a time using parameters like maxSurge (how many new Pods to add temporarily) and maxUnavailable (how many old Pods can be down during the update).
- Recreate is a simpler but more disruptive strategy. When we use Recreate, K8s first stops and deletes all existing Pods, then starts new ones with the updated configuration. This approach causes downtime, because there's a moment when no Pods are running.

It's sometimes used in scenarios where the new version of the app can't run alongside the old one—for example, if they both use the same resource like a database lock or local storage path.

Test Rollout strategies in Deployment:

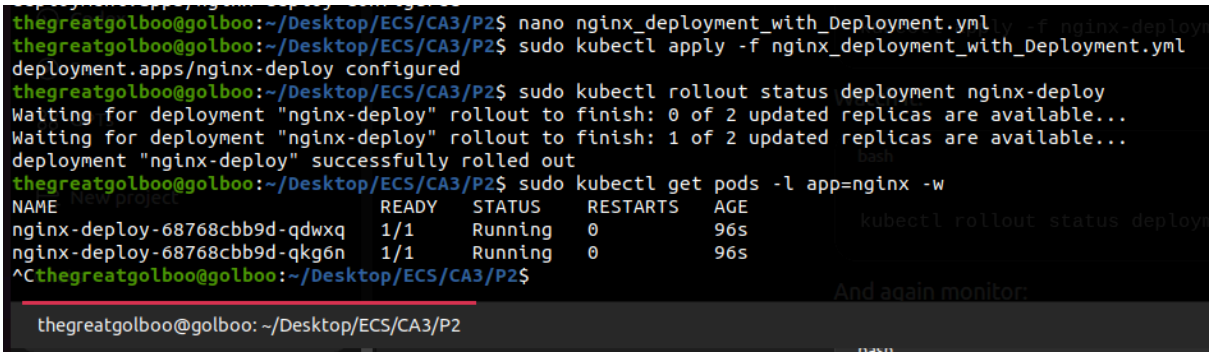
We use the our prev yaml file(in prev section).

To test RollingUpdate strategy, we set strategy type to RollingUpdate.



we'll observe:New Pods with the updated version being created one-by-one, Old Pods being terminated gradually, app always stays available (at least 2 Pods running at any time)

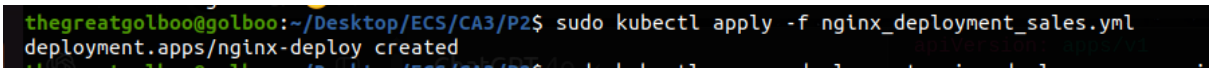
To test Recreate strategy, we set strategy type to Recreate.



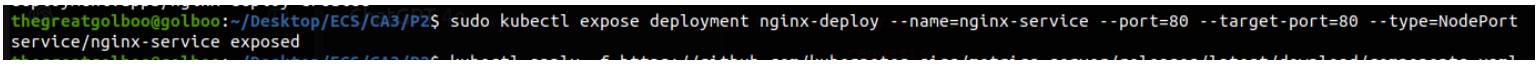
we'll see: All old Pods are deleted at once, Then new Pods are created ,There's a short downtime during the switch (no Pods run briefly)

3. Sales campaigns, Black Friday, holiday nights, and …

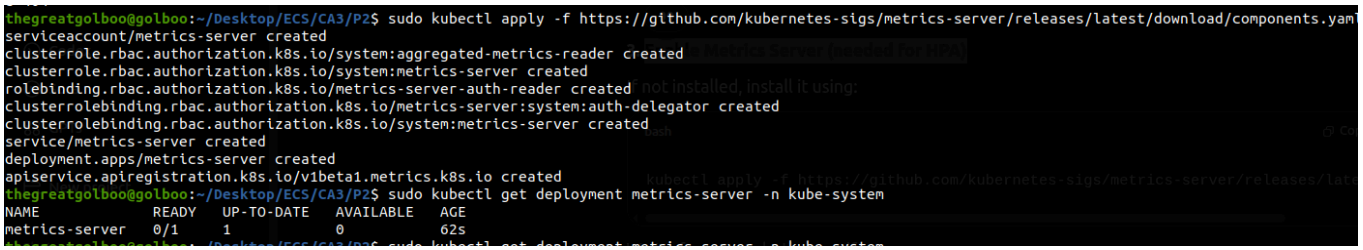
We created a Deployment with 5 NGINX Pods and the then applied it:



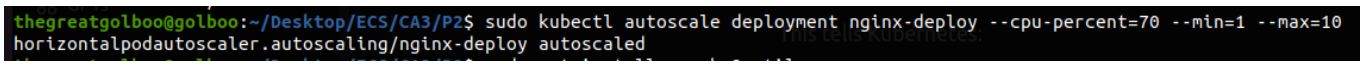
Then, we exposed the deployment via a service:



then , we enabled metrics server (needed for HPA):

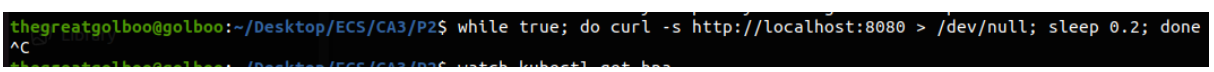


After that, we create a Horizontal Pod Autoscaler (HPA):

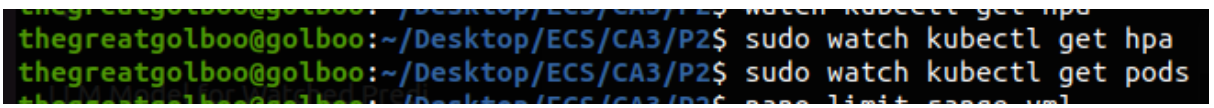


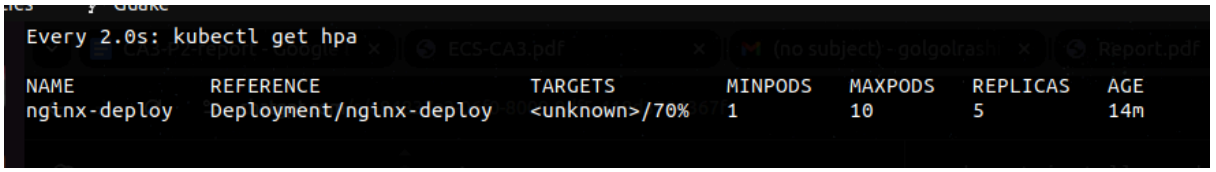
- If average CPU usage exceeds 70%, increase the number of pods (up to 10)
- If usage drops, scale down to as few as 1

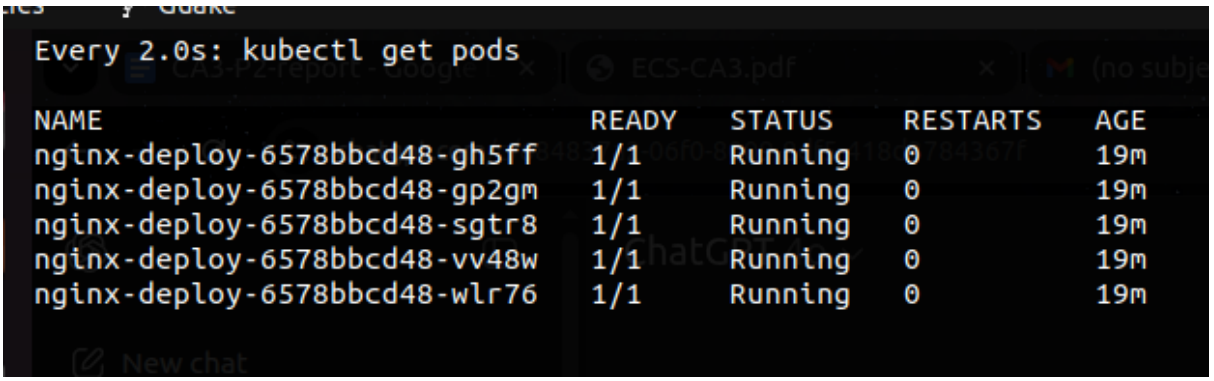
We ran a simple loop to simulate repeated traffic:



We ran these to watch pod count change and see the updated pod count:







Questions:

1. در کوبرنتیز، زمانی که برنامه‌ای به فضای ذخیره‌سازی نیاز دارد، ما نمی‌توانیم به‌صورت مستقیم یک مسیر از دیسک را به آن اختصاص دهیم. به جای آن از مفاهیمی مثل PV، PVC و StorageClass استفاده می‌شود تا این فرایند به‌صورت استاندارد، پویا و مدیریت‌شده انجام شود.

- PV یا PersistentVolume در واقع یک منبع ذخیره‌سازی است که توسط ادمین تعریف شده و در اختیار کلاستر قرار می‌گیرد. این منبع می‌تواند روی یک دیسک محلی، NFS، یا سرویس‌های ابری مثل AWS EBS، Google Persistent Disk و ... باشد.
- PVC یا PersistentVolumeClaim درخواست برنامه یا کاربر برای استفاده از فضای ذخیره‌سازی است. کاربر در PVC مشخص می‌کند که چه مقدار فضا لازم دارد و چه نوع ویژگی‌هایی موردنیاز است (مثلاً سرعت، خواندن/نوشتن هم‌زمان و...).
- StorageClass به کوبرنتیز کمک می‌کند تا مشخص کند چه نوع دیسکی ساخته شود. مثلاً اگر بخواهیم حجم‌هایی با سرعت بالا، یا SSD، یا رمزگذاری‌شده داشته باشیم، StorageClass تعیین‌کننده این ویژگی‌ها است.

این اجزا با هم باعث می‌شوند فضای ذخیره‌سازی در کوبرنتیز به صورت داینامیک، اتوماتیک و قابل تنظیم مدیریت شود و از وابستگی مستقیم برنامه‌ها به سخت‌افزار جلوگیری شود.

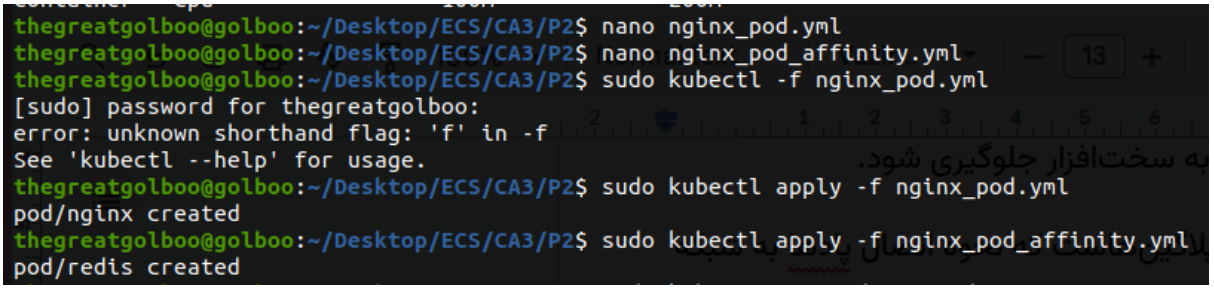
2. CNI یا Container Network Interface مجموعه‌ای از پلاگین‌هاست که نحوه اتصال پادها به شبکه را مشخص می‌کند. کوبرنتیز به تنهایی شبکه‌سازی نمی‌کند؛ بلکه از این پلاگین‌ها استفاده می‌کند تا ارتباط بین پادها، نودها و خارج از کلاستر برقرار شود.

Calico یکی از معروف‌ترین CNIهاست. از روش IP Routing برای ارتباط استفاده می‌کند. این یعنی هر پاد یک IP دارد و ترافیک مستقیماً بین IPها مسیریابی می‌شود. مزیت Calico در سادگی، عملکرد خوب در مقیاس بالا و پشتیبانی از سیاست‌های امنیتی (Network Policies) است.

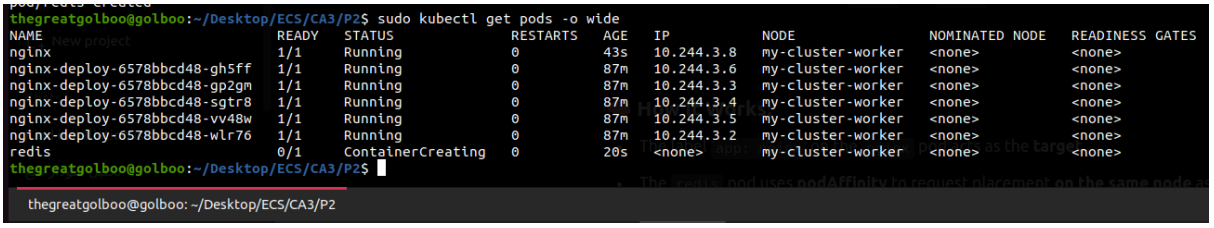
Cilium یک CNI مدرن‌تر است که از فناوری eBPF در هسته لینوکس استفاده می‌کند. برخلاف Calico که بیشتر به مدل‌های سنتی تکیه دارد، Cilium امکان نظارت بهتر، امنیت دقیق‌تر، و کارایی بیشتر را فراهم می‌کند. برای مثال، می‌توان با آن کنترل دقیق‌تری روی ترافیک ورودی و خروجی پادها اعمال کرد

.3

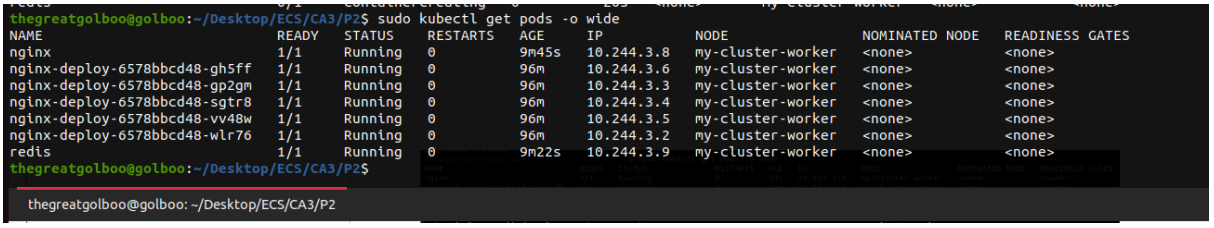
First, deploy the nginx pod with a label, also we need redis pod with Pod Affinity toward nginx. After creating yml files, we apply them:



We need to verify if both pods are on the same node:



The nginx pod is scheduled on node: my-cluster-worke. Alsothe redis pod is also being scheduled (currently ContainerCreating) on the same node: my-cluster-worker



4. KEDA ابزاری است که به Kubernetes این توانایی را می‌دهد تا برنامه‌ها را نه‌تنها بر اساس مصرف منابعی مثل CPU و RAM، بلکه بر اساس رویدادهای خارجی مقیاس‌پذیر کند. به‌عبارت ساده‌تر، KEDA این امکان را فراهم می‌کند که اگر مثلاً تعداد پیام‌های داخل یک صف مثل Kafka یا RabbitMQ زیاد شود، به‌صورت خودکار پادهایی ایجاد شوند که این پیام‌ها را پردازش کنند، و وقتی حجم کار کاهش پیدا کرد، این پادها حذف شوند، حتی تا حدی که به صفر برسند. این ویژگی، KEDA را به ابزاری بسیار کاربردی برای سناریوهایی تبدیل کرده که در آن‌ها بار کاری یکنواخت نیست و ممکن است در ساعات یا روزهایی از هفته، تقاضا به‌شدت افزایش یا کاهش یابد؛ مثل سیستم‌های فروش، اپ‌های پیام‌رسان، یا سرویس‌هایی که بر اساس رخدادهای بیرونی کار می‌کنند. به‌جای اینکه همیشه چند پاد فعال بمانند و منابع مصرف کنند، KEDA کمک می‌کند تنها زمانی که واقعاً نیاز هست پاد ایجاد شود؛ این یعنی کاهش مصرف منابع و در نتیجه کاهش هزینه، بدون اینکه سرویس‌دهی به کاربر دچار اختلال شود. از آن‌جایی که KEDA از منابع مختلفی پشتیبانی می‌کند — مثل Azure Queue، AWS SQS، Redis، Kafka و حتی Prometheus — می‌توان گفت که انعطاف‌پذیری بسیار بالایی دارد و به‌خوبی با زیرساخت‌های متنوع هماهنگ می‌شود. این قابلیت باعث شده KEDA یکی از بهترین انتخاب‌ها برای اپلیکیشن‌هایی باشد که معماری آن‌ها بر اساس رخداد یا صف طراحی شده است.

Source: Workshop

AI: ChatGPT- can’t provide link because of sharing images

Prompts:

- i am trying to install kind in ubuntu. this image is the commands and outputs. fix it.
- thegreatgolboo@golboo:~/Desktop/ECS/CA3/P2$ kind create cluster --name myCluster --config=./cluster_config.yml ERROR: failed to create cluster: failed to list nodes: command "docker ps -a --filter label=io.x-k8s.kind.cluster=myCluster --format '{{.Names}}'" failed with error: exit status 1 Command Output: permission

denied while trying to connect to the Docker daemon socket at unix:///var/run/docker.sock: Get "http://%2Fvar%2Frun%2Fdocker.sock/v1.47/containers/json?all=1&filters=%7B%22label%22%3A%7B%22io.x-k8s.kind.cluster%3DmyCluster%22%3Atrue%7D%7D": dial unix /var/run/docker.sock: connect: permission denied why?

- this is my cluster config file. the command in the second image seems taking so long. why? how to fix?
- how to verify?
- thegreatgolboo@golboo:~/Desktop/ECS/CA3/P2$ kubectl cluster-info --context kind-my-cluster Command 'kubectl' not found, but can be installed with: sudo snap install kubectl thegreatgolboo@golboo:~/Desktop/ECS/CA3/P2$ sudo snap install kubectl [sudo] password for thegreatgolboo: error: This revision of snap "kubectl" was published using classic confinement and thus may perform arbitrary system changes outside of the security sandbox that snaps are usually confined to, which may put your system at risk. If you understand and want to proceed repeat the command including --classic.
- $ kubectl cluster-info --context kind-my-cluster error: context "kind-my-cluster" does not exist
- how to apply this naminefest. also can i use another image instead of alpine? for example the image i loaded
- can i use this image that was loaded with thw below command? docker pull kindest/node:v1.29.2
- which is lighter to use?
- thegreatgolboo@golboo:~/Desktop/ECS/CA3/P2$ sudo kubectl apply -f RCs_with_same_selector.yml [sudo] password for thegreatgolboo: error: error validating "RCs_with_same_selector.yml": error validating data: failed to download openapi: Get "https://127.0.0.1:37853/openapi/v2?timeout=32s": dial tcp 127.0.0.1:37853: connect: connection refused; if you choose to ignore these errors, turn validation off with --validate=false fix this error
- thegreatgolboo@golboo:~/Desktop/ECS/CA3/P2$ sudo kubectl cluster-info --context kind-my-cluster E0611 17:53:43.770767 14882 memcache.go:265]

"Unhandled Error" err="couldn't get current server API group list: Get \"https://127.0.0.1:37853/api?timeout=32s\": dial tcp 127.0.0.1:37853: connect: connection refused" E0611 17:53:43.772301 14882 memcache.go:265] "Unhandled Error" err="couldn't get current server API group list: Get \"https://127.0.0.1:37853/api?timeout=32s\": dial tcp 127.0.0.1:37853: connect: connection refused" E0611 17:53:43.773867 14882 memcache.go:265] "Unhandled Error" err="couldn't get current server API group list: Get \"https://127.0.0.1:37853/api?timeout=32s\": dial tcp 127.0.0.1:37853: connect: connection refused" E0611 17:53:43.775173 14882 memcache.go:265] "Unhandled Error" err="couldn't get current server API group list: Get \"https://127.0.0.1:37853/api?timeout=32s\": dial tcp 127.0.0.1:37853: connect: connection refused" E0611 17:53:43.776821 14882 memcache.go:265] "Unhandled Error" err="couldn't get current server API group list: Get \"https://127.0.0.1:37853/api?timeout=32s\": dial tcp 127.0.0.1:37853: connect: connection refused" To further debug and diagnose cluster problems, use 'kubectl cluster-info dump'. The connection to the server 127.0.0.1:37853 was refused - did you specify the right host or port?

- use ReplicationControlle?
- why? ~/Desktop/ECS/CA3/P2$ sudo kubectl scale rc nginx-rc-v1 --replicas=1 error: no objects passed to scale replicationcontrollers "nginx-rc-v1" not found
- but: thegreatgolboo@golboo:~/Desktop/ECS/CA3/P2$ sudo kubectl apply -f nginx_deployment_with_RC_v1.yml deployment.apps/nginx-deploy unchanged thegreatgolboo@golboo:~/Desktop/ECS/CA3/P2$ sudo kubectl get rc NAME DESIRED CURRENT READY AGE nginx-rc-one 1 1 1 110m nginx-rc-two 1 1 1 110m nginx-rc-v2 1 1 1 4m32s thegreatgolboo@golboo:~/Desktop/ECS/CA3/P2$
- i have applied nginx-rc-v1 !
- how to check the scaled number?
- How to use ReplicaSet
- what about using deployment
- how to delete my prev applys to cluster when i was trying reblicacontroller
- Rollout strategy in deployment

- differnece of deployment and replicaset
- How to test the RollingUpdate and Recreate on the application manifest
- thegreatgolboo@golboo:~/Desktop/ECS/CA3/P2$ sudo kubectl apply -f nginx_deployment_with_Deployment.yml [sudo] password for thegreatgolboo: error: error validating "nginx_deployment_with_Deployment.yml": error validating data: failed to download openapi: Get "https://127.0.0.1:40019/openapi/v2?timeout=32s": EOF; if you choose to ignore these errors, turn validation off with --validate=false
- kubectl get pods -l app=nginx -w why above command is not ending
- ~/Desktop/ECS/CA3/P2$ sudo kubectl apply -f nginx_deployment_with_Deployment.yml The Deployment "nginx-deploy" is invalid: spec.strategy.rollingUpdate: Forbidden: may not be specified when strategy `type` is 'Recreate'
- apiVersion: apps/v1 kind: Deployment metadata: name: nginx-deploy spec: replicas: 2 selector: matchLabels: app: nginx strategy: type: Recreate rollingUpdate: maxUnavailable: 1 maxSurge: 1 template: metadata: labels: app: nginx spec: containers: - name: nginx image: nginx:1.22 ports: - containerPort: 80 this is my manifest but iam still gettiing that error
- apiVersion: apps/v1 kind: Deployment metadata: name: nginx-deploy spec: replicas: 5 selector: matchLabels: app: nginx template: metadata: labels: app: nginx spec: containers: - name: nginx image: nginx ports: - containerPort: 80 resources: requests: cpu: "100m" limits: cpu: "200m" apply limit range for above manifest
- limit range for my manifest should be in another yml file?
- thegreatgolboo@golboo:~/Desktop/ECS/CA3/P2$ kubectl apply -f https://github.com/kubernetes-sigs/metrics-server/releases/latest/download/components.yaml error: error validating "https://github.com/kubernetes-sigs/metrics-server/releases/latest/download/components.yaml": error validating data: failed to download openapi: Get "http://localhost:8080/openapi/v2?timeout=32s": dial tcp 127.0.0.1:8080: connect: connection refused; if you choose to ignore these errors, turn validation off with --validate=false

- thegreatgolboo@golboo:~/Desktop/ECS/CA3/P2$ ab -n 100000 -c 100 http://127.0.0.1:31710/ This is ApacheBench, Version 2.3 <$Revision: 1879490 $> Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/ Licensed to The Apache Software Foundation, http://www.apache.org/ Benchmarking 127.0.0.1 (be patient) apr_socket_recv: Connection refused (111)
- thegreatgolboo@golboo:~/Desktop/ECS/CA3/P2$ kubectl port-forward service/nginx-service 8080:80 E0612 00:38:11.197570 59174 memcache.go:265] "Unhandled Error" err="couldn't get current server API group list: Get \"http://localhost:8080/api?timeout=32s\": dial tcp 127.0.0.1:8080: connect: connection refused" E0612 00:38:11.200343 59174 memcache.go:265] "Unhandled Error" err="couldn't get current server API group list: Get \"http://localhost:8080/api?timeout=32s\": dial tcp 127.0.0.1:8080: connect: connection refused" E0612 00:38:11.204428 59174 memcache.go:265] "Unhandled Error" err="couldn't get current server API group list: Get \"http://localhost:8080/api?timeout=32s\": dial tcp 127.0.0.1:8080: connect: connection refused" E0612 00:38:11.208075 59174 memcache.go:265] "Unhandled Error" err="couldn't get current server API group list: Get \"http://localhost:8080/api?timeout=32s\": dial tcp 127.0.0.1:8080: connect: connection refused" The connection to the server localhost:8080 was refused - did you specify the right host or port?
- anoyher way for creating trafic?
- how to test and verify applying limit range
- want to do this part
- this is the result. is it right?
- Explain what is KEDA