LinuxFoundation.CKA.v2025-09-01.q86

試験コード:	CKA
試験名称:	Certified Kubernetes Administrator (CKA) Program Exam
認定資格:	Linux Foundation
無料問題数:	86
バージョン:	v2025-09-01
アクセス数:	110
ページビュー数:	860
https://www.jpnpdf.com/LinuxFoundation.CKA.v2025-09-01.q86-mondaishu.html	

最新問題: **1**
ストレージにPersistentVolumeClaimを使用するポッドがあります。ポッドは削除されましたが、ボリューム上のデータは依然として残っています。データが削除されない理由と、この動作を変更する方法について説明してください。
Answer:
以下のステップバイステップの説明付きの解決策を参照してください。
Explanation:
解決策（ステップバイステップ）：
ポッドが削除されても、ボリューム上のデータはデフォルトでは自動的に削除されません。これは、PersistentVolume の persistentVolumeReclaimPolicy がデフォルトで「Retain」に設定されているためです。このポリシーにより、ポッドが削除されてもボリュームは削除されず、データが保持されます。
ポッドの削除時にデータを削除するには、「persistentVolumeReclaimPolicy」を「Delete」に変更する必要があります。手順は以下のとおりです。
1. PersistentVolume を更新します。
- PersistentVolume 定義で 'persistentVolumeReclaimPolicy' を 'Delete' に更新します。

2. 変更を適用します。- 'kubectl apply -f my-pv.yaml' を使用して、更新された PersistentVolume 定義を適用します。これで、この PersistentVolume を使用しているポッドが削除されると、ボリュームとデータも自動的に削除されます。

最新問題: **2**
ポッド ラベル name=cpu-utilizer から、CPU ワークロードの高いポッドを見つけ、最も多くの CPU を消費しているポッドの名前をファイル /opt/KUTR00102/KUTR00102.txt (既に存在) に書き込みます。
Answer:
以下の解決策を参照してください。
説明
解決
F:\Work\Data Entry Work\Data Entry\20200827\CKA\16 B.JPG

F:\Work\Data Entry Work\Data Entry\20200827\CKA\16 C.JPG

最新問題: **3**

pod foo のログを監視し、次の操作を実行します。

* アクセスできないウェブサイトに対応するログ行を抽出

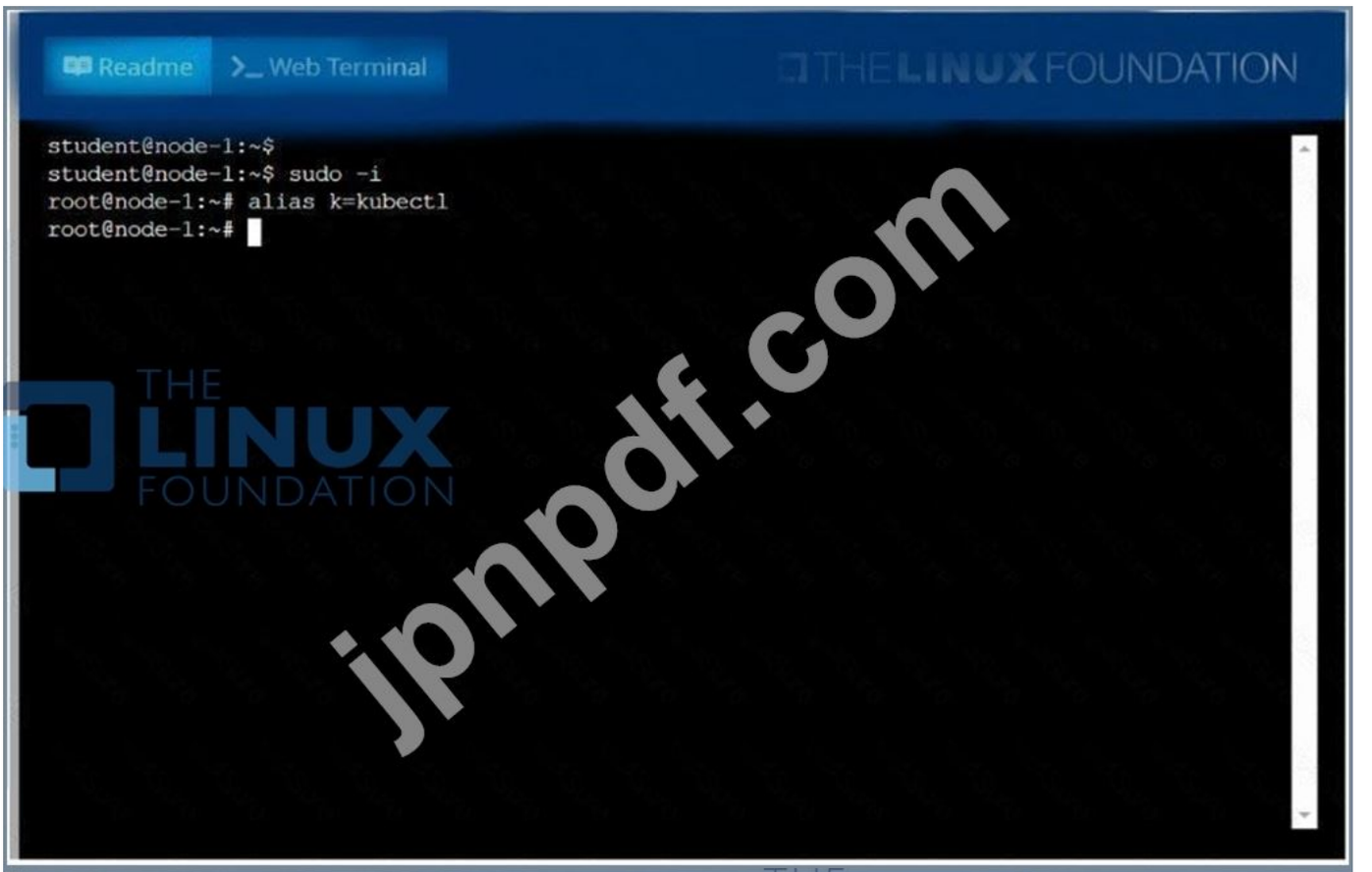
* /opt/KULM00201/foo に書き込む

Answer:

以下の解決策を参照してください。

説明

解決



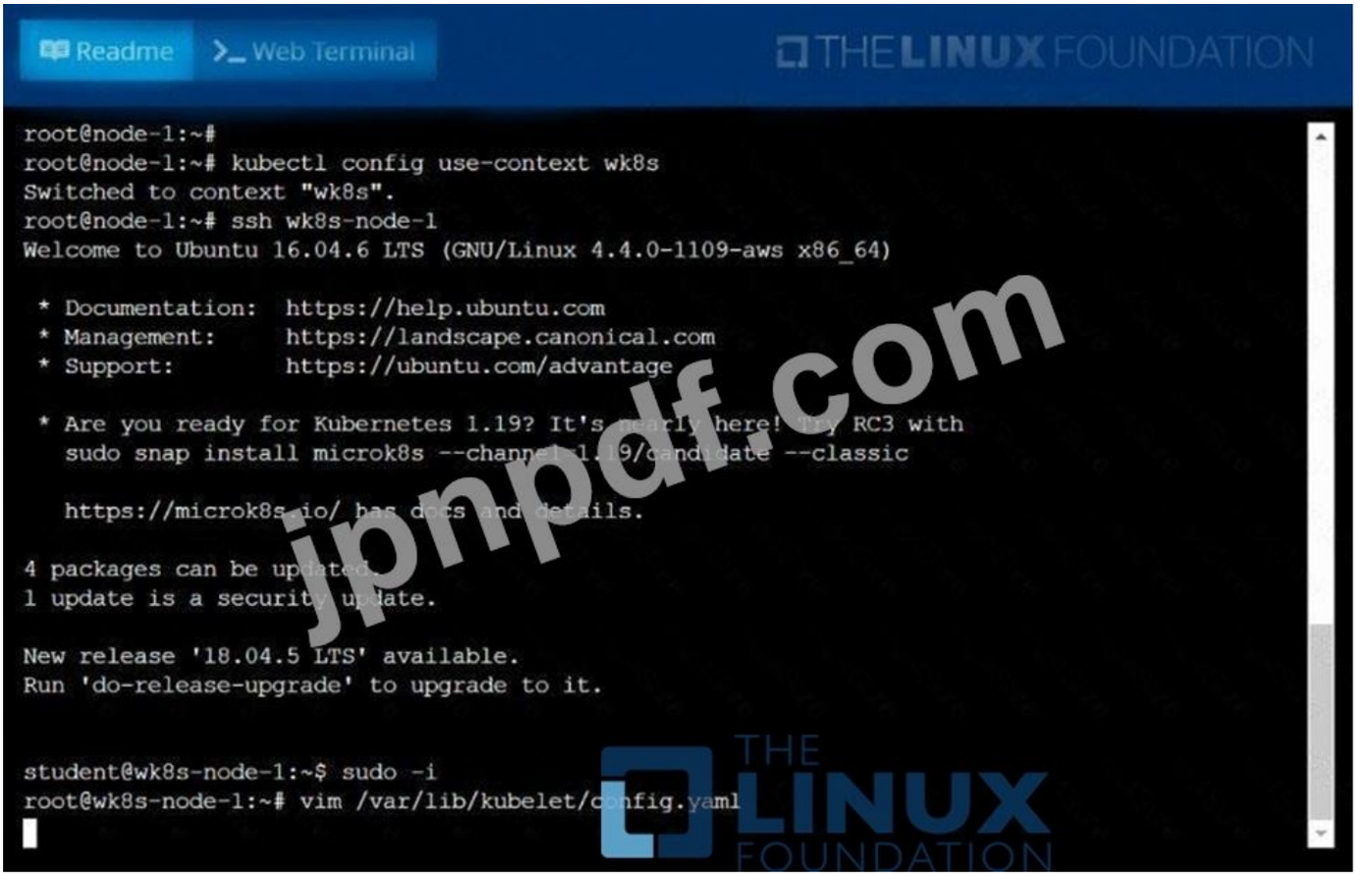
最新問題: **4**

作成したデプロイメントとhpaを削除してクラスターをクリーンアップします。

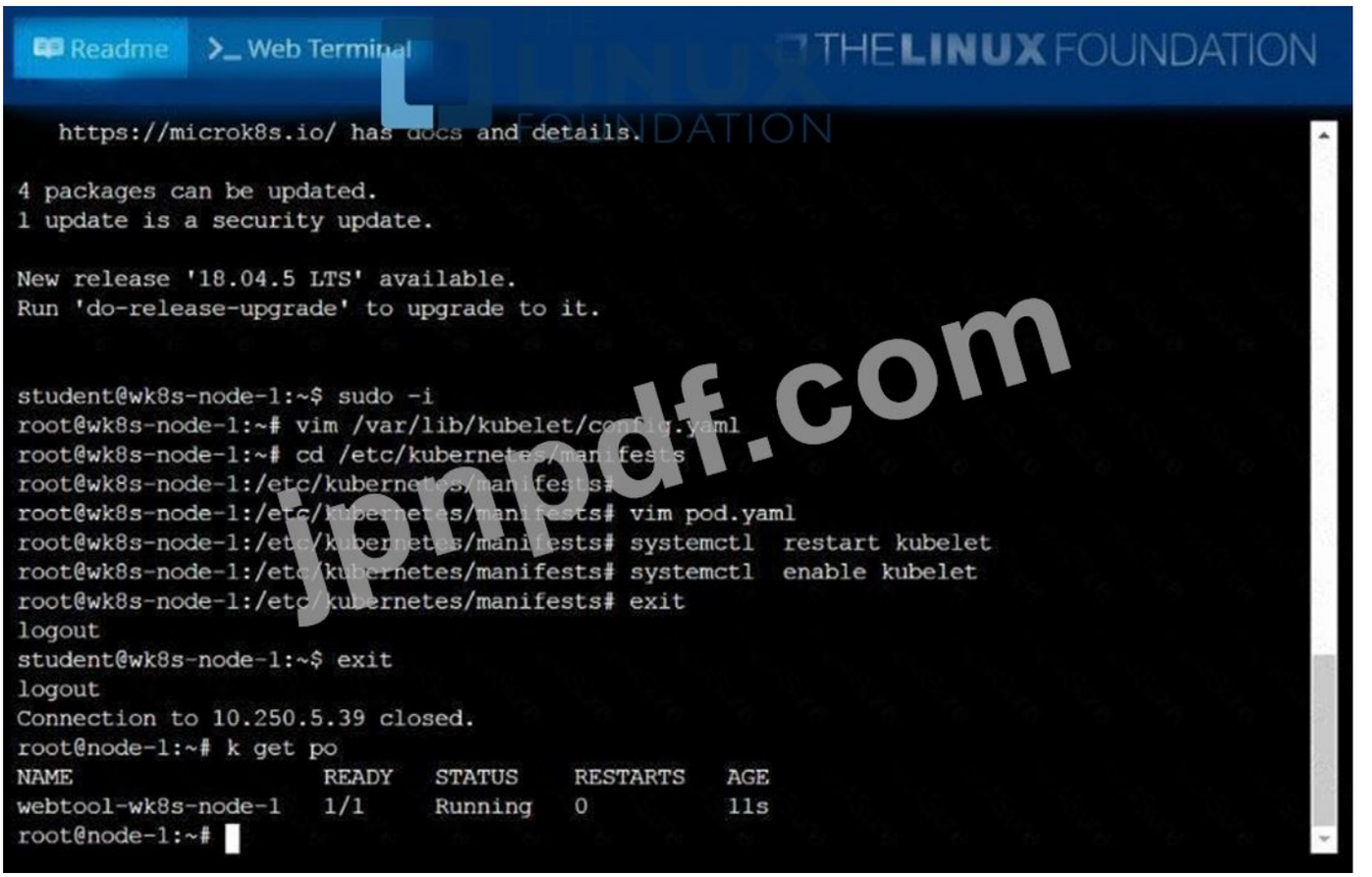
Answer:

kubectl でデプロイしたWebアプリを削除し、kubectl で hpa Webアプリを削除

最新問題: **5**
name=wk8s-node-1 というラベルのノード上で、kubelet systemd 管理サービスを設定し、webtool という名前の Image httpd コンテナを 1 つ含むポッドを自動的に起動します。必要な仕様ファイルは、ノードの /etc/kubernetes/manifests ディレクトリに配置する必要があります。
次のコマンドを使用して適切なノードに SSH 接続できます。
[student@node-1] $ ssh wk8s-node-1
次のコマンドを使用して、ノード上で昇格された権限を取得できます。
[student@wk8s-node-1] $ | sudo -i
Answer:
以下の解決策を参照してください。
説明
解決




Readme
Web Terminal
THE LINUX FOUNDATION
clientCAFile: /etc/kubernetes/pki/ca.crt
authorization:
mode: Webhook
webhook:
cacheAuthorizedTTL: 0s
cacheUnauthorizedTTL: 0s
clusterDNS:
- 10.96.0.10
clusterDomain: cluster.local
cpuManagerReconcilePeriod: 0s
evictionPressureTransitionPeriod: 0s
fileCheckFrequency: 0s
healthzBindAddress: 127.0.0.1
healthzPort: 10248
httpCheckFrequency: 0s
imageMinimumGCAge: 0s
kind: KubeletConfiguration
rotateCertificates: true
runtimeRequestTimeout: 0s
staticPodPath: /etc/kubernetes/manifests
syncFrequency: 0s
:wq
jpnpdf.com
Readme
Web Terminal
THE LINUX FOUNDATION
root@node-1:~# ssh wk8s-node-1
Welcome to Ubuntu 16.04.6 LTS (GNU/Linux 4.4.0-1109-aws x86_64)
* Documentation: https://help.ubuntu.com
* Management: https://landscape.canonical.com
* Support: https://ubuntu.com/advantage
* Are you ready for Kubernetes 1.19? It's nearly here! Try RC3 with
sudo snap install microk8s --channel=1.19/candidate --classic
https://microk8s.io/ has docs and details.
4 packages can be updated.
1 update is a security update.
New release '18.04.5 LTS' available.
Run 'do-release-upgrade' to upgrade to it.
student@wk8s-node-1:~$ sudo -i
root@wk8s-node-1:~# vim /var/lib/kubelet/config.yaml
root@wk8s-node-1:~# cd /etc/kubernetes/manifests
root@wk8s-node-1:/etc/kubernetes/manifests#
root@wk8s-node-1:/etc/kubernetes/manifests# vim pod.yaml
jpnpdf.com



最新問題: **6**
スコア: 7%

タスク
既存のデプロイメント フロントエンドを再構成し、既存のコンテナ nginx のポート 80/tcp を公開する http という名前のポート仕様を追加します。
コンテナ ポート http を公開する front-end-svc という名前の新しいサービスを作成します。
新しいサービスを設定して、個々のポッドがスケジュールされているノード上の NodePort 経由でそれらのポッドも公開するようにします。
Answer:
解決：
kubectl get deploy フロントエンド
kubectl edit deploy フロントエンド -o yaml
#http という名前のポート指定
#サービス.yaml
APIバージョン: v1

種類: サービス
メタデータ:
名前: フロントエンドサービス
ラベル:
アプリ: nginx
仕様:
ポート:
- ポート: 80
プロトコル: TCP
名前: http
セレクタ：
アプリ: nginx
タイプ: NodePort
kubectl create -f service.yaml
kubectl get svc
http という名前のポート指定
kubectl expose デプロイメント フロントエンド --name=front-end-svc --port=80 --tarport=80 --type=NodePort

最新問題**: 7**
少なくとも 3Gi のストレージとアクセス モードが ReadWriteOnce の PersistentVolumeClaim を作成し、ステータスが Bound であることを確認します。
A. vim タスク-pv-claim.yaml
APIバージョン: v2
種類: PersistentVolumeClaim
メタデータ:
名前: タスクPVクレーム
仕様:
ストレージクラス名: ""
アクセスモード:
- 一度だけ読み書き可能
リソース：
リクエスト:
ストレージ: 4Gi
kubectl apply -f タスクpv-claim.yaml
//確認する
kubectl でPVを取得する
名前 容量 アクセス
モード クレーム ポリシー ステータス クレーム
ストレージクラス理由年齢

タスクPVボリューム 4Gi RWO
バインドされたデフォルト/タスクPVクレームを保持する
6分16秒
kubectl で PVC を取得する
名前 ステータス ボリューム
容量 アクセスモード ストレージクラス 年齢
task-pv-claim バインドされた task-pv-volume
5彼らはRWO 6s
B. vim タスク-pv-claim.yaml
APIバージョン: v1
種類: PersistentVolumeClaim
メタデータ:
名前: タスクPVクレーム
仕様:
ストレージクラス名: ""
アクセスモード:
- 一度だけ読み書き可能
リソース：
リクエスト:
ストレージ: 3Gi
kubectl apply -f タスクpv-claim.yaml
//確認する
kubectl でPVを取得する
名前 容量 アクセス
モード クレーム ポリシー ステータス クレーム
ストレージクラス理由年齢
タスクPVボリューム5Gi RWO
バインドされたデフォルト/タスクPVクレームを保持する
6分16秒
kubectl で PVC を取得する
名前 ステータス ボリューム
容量 アクセスモード ストレージクラス 年齢
task-pv-claim バインドされた task-pv-volume
5彼らはRWO 6s
Answer: (解答を表示する**)**

最新問題**: 8**
複数のノードにまたがって複数のポッドがスケジュールされているKubernetesクラスターを管理しています。デプロイメントの1つ（「wordpress」）では、「WordpressConfig」というカスタムリソース定義（CRD）を使用しており、Wordpressアプリケーションの構成を定義しています。

「WordpressConfig」CRDには、データベース接続、セキュリティ設定、その他のアプリケーション固有の設定に関するパラメータが含まれています。WordPressアプリケーションを実行しているポッドが、ポッドが既に実行中であっても、「WordpressConfig」CRDの最新の構成変更を確実に認識できるようにしたいと考えています。

Answer:

以下のステップバイステップの説明付きの解決策を参照してください。

Explanation:

解決策（ステップバイステップ）：

1. ConfigMapを作成します。

-現在の設定を含む「wordpress-config」という名前のConfigMapを定義します。「WordpressConfig」CRD。

- 「kubectl get wordpressconfig -o yaml」コマンドを使用して、現在の設定を取得します。「WordpressConfig」リソース。

- 必要な設定データでConfigMapを更新します。これは以下の手順で実行できます。

- 構成データを使用して ConfigMap を手動で作成します。

- 「WordpressConfig」CRD からデータを抽出し、ConfigMap に適用するための別のスクリプトまたは関数を作成します。

- 「wordpress-config」ConfigMap の例として、次の YAML スニペットを使用できます。

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: wordpress-config
data:
  database_host: "your-database-host"
  database_port: "your-database-port"
  database_user: "your-database-user"
  database_password: "your-database-password"
  # ...other configuration parameters
```

2. Wordpress のデプロイメントを更新します。- 「wordpress-config」ConfigMap をボリュームとして使用するように「wordpress」デプロイメントを変更します。- ボリュームをポッドのコンテナにマウントし、アプリケーションがボリュームから構成にアクセスできることを確認します。- デプロイメント YAML を次のように更新します。

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: wordpress
spec:
  template:
    spec:
      containers:
      - name: wordpress
        image: your-wordpress-image:latest
        volumeMounts:
          - name: config-volume
            mountPath: "/etc/wordpress"
      volumes:
        - name: config-volume
          configMap:
            name: wordpress-config
```

3. コントローラーの実装 :- 「WordpressConfig」CRDの変更を監視するカスタムコントローラーを作成します。- 変更が発生すると、コントローラーは「wordpress-config」ConfigMapを新しい構成データで更新する必要があります。- これにより、ConfigMapが「WordpressConfig」リソースと同期した状態を維持できます。- コントローラーは、Go、Python、Javaなど、さまざまな言語で記述できます。4. ポッドの更新を構成する :- ConfigMapの変更後にWordPressポッドが自動的に再起動されない場合は、「kubectl rollout restart deployment wordpress」コマンドを使用して再起動をトリガーできます。- または、デプロイメント構成で「imagePullPolicy」を設定すると、ポッドは更新ごとに新しいイメージをプルし、構成を再適用するように強制されます。5. 構成を確認する :- これらの手順を実装したら、WordpressポッドがConfigMapから最新の構成を使用していることを確認します。- 'kubectl get pods -l app=wordpress -o yamp' コマンドを使用して、実行中のポッドを調べ、ボリュームマウントと設定データのパスを確認します。- 実行中のWordPressアプリケーションにアクセスし、新しい設定が適用されていることを確認します。これらの手順に従うことで、ポッドが既に実行されているかどうかに関係なく、Wordpressポッドが常に「WordpressConfig」CRDの最新の設定を使用するようになります。

最新問題: **9**
準備ができているノードをチェックし、ファイル /opt/nodestatus に出力します。
A. JSONPATH='{範囲 .items[*]}{@.metadata.name}:{範囲
@.status.conditions[*]}{@.type}={@.status};{end}{end}' \
&& kubectl get nodes -o jsonpath="$JSONPATH" | grep
「準備完了=True」> /opt/node-status
//確認する
cat /opt/node-status
B. JSONPATH='{範囲 .items[*]}{@.metadata.name}:{範囲
@.status.conditions[*]}{@.type}={@.status};{end}{end}' \
//確認する
cat /opt/node-status
Answer: A (メーッセージを残す**)**

最新問題: **10**
スコア: 4%

タスク
ek8s-node-1 という名前のノードを使用不可に設定し、そのノードで実行されているすべてのポッドを再スケジュールします。
Answer:
解決：
[student@node-1] > ssh ek8s
kubectl コルドン ek8s-node-1
kubectl ドレイン ek8s-node-1 --delete-local-data --ignore-daemonsets --force

最新問題: **11**
非常に特殊な RBAC セットアップを使用して Kubernetes クラスターを管理しています。
- 「engineering」グループのユーザーは、「dev」名前空間にポッドを作成できますが、ポッドの名前が「frontend」で始まる場合に限られます。
- 「security」グループのユーザーは、「prod」名前空間内のイベントを表示できますが、イベントの理由が「失敗」の場合のみです。
YAML ファイルを作成してカスタム リソースを定義し、「エンジニアリング」グループと「セキュリティ」グループの権限を定義します。
Answer:
以下のステップバイステップの説明付きの解決策を参照してください。
Explanation:
解決策（ステップバイステップ）：
1 「dev」および「prod」名前空間を作成します。
kubectl で名前空間 dev を作成する
kubectl で名前空間 prod を作成する
2. 「pod-events」のカスタム リソース定義 (CRD) を定義します。

```
apiVersion: apiextensions.k8s.io/v1
kind: CustomResourceDefinition
metadata:
  name: pod-events.example.com
spec:
  group: example.com
  versions:
  - name: v1
    served: true
    storage: true
  scope: Namespaced
  names:
    plural: pod-events
    singular: podevent
```

3. 「engineering-pod-creation」ロールを定義します。

```
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: engineering-pod-creation
  namespace: dev
rules:
- apiGroups: ["apps"]
  resources: ["pods"]
  verbs: ["create"]
  resourceNames: ["frontend"]
```

4. 「engineering-pod-creation」ロールバインディングを作成します。

```
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: engineering-pod-creation-binding
  namespace: dev
subjects:
- kind: Group
  name: engineering
  apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: Role
  name: engineering-pod-creation
  apiGroup: rbac.authorization.k8s.io
```

5. 「security-event-view」ロールを定義します。

```
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: security-event-view
  namespace: prod
rules:
- apiGroups: ["example.com"]
  resources: ["pod-events"]
  verbs: ["get", "list", "watch"]
  resourceNames: ["Failed"]
```

6. 「security-event-view」ロールバインディングを作成します。

7. YAMLファイルをクラスターに適用します: kubectl apply -f rbac-config.yaml

最新問題**: 12**

次のようにポッドを作成します。

* 名前: mongo

* 使用画像: mongo

* 新しいKubernetes名前空間に「my-website」という名前を付けます

Answer:

最新問題**: 13**
ポート 30080 で Web アプリケーションを公開する NodePort サービスを持つ Kubernetes クラスターがあります。NetworkPolicy を使用して、特定の IP アドレス (192.168.1.10 および 10.0.0.1) からのこのサービスへのアクセスを制限する必要があります。
Answer:
以下のステップバイステップの説明付きの解決策を参照してください。
Explanation:
この制限を適用するために必要な NetworkPolicy を作成します。
解決策（ステップバイステップ）：
ステップ 1: Web アプリケーション サービスへのアクセスを制限する NetworkPolicy を作成します。

```yaml
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: web-app-access-restriction
  namespace: default
spec:
  podSelector: {}
  ingress:
  - from:
    - ipBlock:
        cidr: 192.168.1.10/32
    - ipBlock:
        cidr: 10.0.0.1/32
  egress:
  - to:
    - ipBlock:
        cidr: 0.0.0.0/0
  policyTypes:
  - Ingress
```


ステップ2 :NetworkPolicyをクラスターに適用します。kubectl apply -f web-app-access-restriction.yaml このNetworkPolicyは、Webアプリケーションサービスへの入力トラフィックを指定されたIPアドレスのみに制限します。「podSelector: {F」は、このポリシーが「default」名前空間内のすべてのポッドに適用されるようにします。「egress」セクションは、ポッドからのすべての送信トラフィックを許可します。これで、指定されたIPアドレスのみがNodePortを通じて公開されるWebアプリケーションサービスにアクセスできるようになります。

最新問題: **14**
3つのノードを持つKubernetesクラスターがあります。Node1とNode2は「default」アベイラビリティゾーンにあり、Node3は「us-east-IA」アベイラビリティゾーンにあります。アベイラビリティゾーンを考慮して、ポッドが3つのノードすべてに分散されるようにする必要があります。
この目標を達成するためにクラスタを構成する方法について説明してください。具体的には、「nodeSelector」や「affinity」を活用して、望ましいノード配置を強制する方法を説明してください。選択したアプローチの根拠を説明してください。
Answer:
以下のステップバイステップの説明付きの解決策を参照してください。
Explanation:
解決策（ステップバイステップ） :
ステップ1: ノードラベルを構成する
各ノードに対応するアベイラビリティーゾーンのラベルを付けます。

Node1 および Node2 (デフォルト」の可用性ゾーン内) の場合:
kubectl ラベル ノード 可用性ゾーン = デフォルト
Node3 アベイラビリティゾーン内）の場合：
kubectl ラベル ノード 可用性ゾーン = us-east-la
ステップ2: ノードセレクターを使用する
目的のアベイラビリティゾーンを指定するには、デプロイメントまたはポッド定義で nodeSelector」を使用します。

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.14.2
        nodeSelector:
          availabilitv-zone: us-east-1a
```


これにより、 nginx-deployment」ラベルのポッドがNode3にのみスケジュールされるようになります。ステップ3 :アフィニティを使用する オプション）よりきめ細かな制御が必要な場合は、 アフィニティ」を使用することもできます。例えば、ポッドを複数のアベイラビリティゾーンに分散させるには、次のようにします。

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.14.2
      affinity:
        podAntiAffinity:
          preferredDuringSchedulingIgnoredDuringExecution:
          - weight: 100
            podAffinityTerm:
              labelSelector:
                matchExpressions:
                - key: availability-zone
                  operator: In
                  values:
                  - default
                  - us-east-1a
              topologyKey: topology.kubernetes.io/zone
```


この構成では、異なるアベイラビリティゾーンにポッドを優先的にスケジュールします。理由：ノードセレクター :ラベルに基づいて特定のノードにポッドを誘導するシンプルなメカニズムを提供します。アフィニティ :ポッドを複数のノード またはアベイラビリティゾーン)に分散させる「podAntiAffinity」や、ポッドが同じノードにスケジュールされるようにする「podAffinity」など、より高度なオプションを提供します。アベイラビリティゾーン :あるゾーンで障害が発生しても、他のゾーンにスケジュールされているポッドには影響が及ばないため、高可用性を実現するためにポッドを複数のゾーンに分散します。

最新問題: **15**

ポッド ラベル name=cpu-utilizer から、CPU ワークロードの高いポッドを見つけ、最も多くの CPU を消費しているポッドの名前をファイル /opt/KUTR00102/KUTR00102.txt (既に存在) に書き込みます。

Answer:

解決

最新問題: **16**
ノード「worker-2」に追加された汚染を削除します
Answer:
kubectl taint nodes worker-2 key:NoSchedule- // 確認すると、「node/worker-2 untainted」というメッセージが表示されます。kubectl get nodes -o customcolumns=NAME:.metadata.name,TAINTS:.spec.taints --no-headers

最新問題: **17**
シークレットを環境変数としてロードする nginx ポッドを作成する
A. // ymlファイルを作成する
kubectl run nginx --image=nginx --restart=Never --dry-run -o yaml > nginx.yml
// 以下にenvセクションを追加して作成します

vim nginx.yaml
APIバージョン: v1
種類: ポッド
メタデータ:
ラベル:
実行: nginx
名前: nginx
仕様:
コンテナ:
- 画像: nginx
名前: nginx
envFrom:
- シークレットRef:
名前: 私の秘密
再起動ポリシー: なし
kubectl apply -f nginx.yaml
//確認する
kubectl exec -it nginx - env
B. // ymlファイルを作成する
kubectl run nginx --image=nginx --restart=Never --dry-run -o
yaml > nginx.yml
// 以下にenvセクションを追加して作成します
vim nginx.yaml
実行: nginx
名前: nginx
仕様:
コンテナ:
- 画像: nginx
名前: nginx
envFrom:
- シークレットRef:
名前: 私の秘密
再起動ポリシー: なし
kubectl apply -f nginx.yaml
//確認する
kubectl exec -it nginx - env
Answer: A (メーッセージを残す**)**

最新問題**: 18**

PostgreSQLデータベースサーバーを実行する「postgres-deployment」というデプロイメントがあります。PostgreSQLサーバーは、「postgres-config」というConfigMapに格納された特定の設定ファイルを使用して設定する必要があります。この設定ファイルには、PostgreSQLスーパーユーザーのパスワードなどの機密情報が含まれています。セキュリティを損なうことなく、この機密情報を安全に保存およびマウントするにはどうすればよいでしょうか？

Answer:

以下のステップバイステップの説明付きの解決策を参照してください。

Explanation:

解決策（ステップバイステップ）：

1. ConfigMapを作成します。

- PostgreSQL設定ファイル（例postgresql.conf）を含む「postgres-config」という名前のConfigMapを作成します。このファイルには、スーパーユーザーのパスワードがプレーンテキストで格納されている可能性があります。「kubectl create configmap」に「--from-file」フラグを指定して、ConfigMapを作成します。

kubectl で postgres-config という設定マップを作成します --from-file=postgresql.conf

2. 機密データにはシークレットを使用する:

- PostgreSQLスーパーユーザーのパスワードを安全に保存するために、'postgres-password'というSecretを作成します。

'--from-literal' フラグを指定した 'kubectl create secret generic':

kubectl でシークレット ジェネリック postgres-password を作成します --from-literal=postgres-password="your_postgres_password"

3. ConfigMapを変更します。

- 'postgres-config' ConfigMapを更新し、'postgresql.conf' 内のプレーンテキストのパスワードをプレースホルダまたは環境変数への参照に置き換えます。これにより、ConfigMap内でパスワードがプレーンテキストで公開されることを防ぎます。

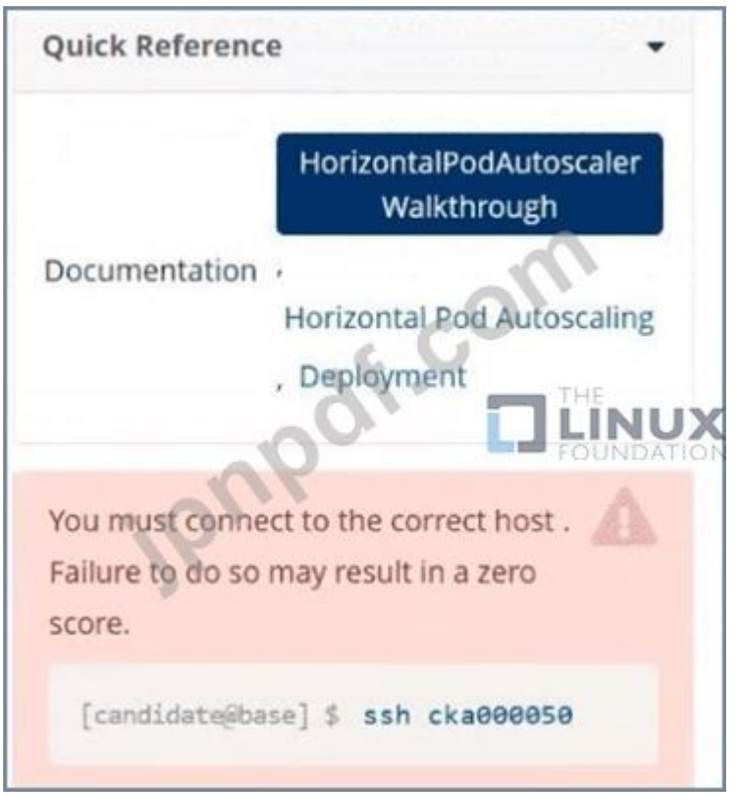
kubectl patch configmap postgres-config -p '{"data": {"postgresql.conf": "password '$POSTGRES パスワード' "}}'

4. デプロイメントを構成する:

- 'postgres-deployment'デプロイメントを変更し、'postgres-config' ConfigMapと'postgres-password' Secretの両方をPodテンプレートのボリュームとしてマウントします。'volumeMounts'でマウントパスを指定し、'volumes'でボリュームソースを定義します。

5. 変更を適用します: - 'kubectl apply -f postgres-deployment.yamr' を使用して、変更したデプロイメント YAML を適用します。6. 構成を確認します: - PostgreSQL インスタンスに接続して認証を試行し、PostgreSQL コンテナが Secret からの安全なパスワードを使用していることを確認します。

最新問題: 19



タスク
autoscale 名前空間に、apache-server という名前の新しい HorizontalPodAutoscaler (HPA) を作成します。この HPA は、autoscale 名前空間内の既存の Deployment apache-server をターゲットにする必要があります。
HPAをポッドあたり50%のCPU使用率を目標に設定し、ポッドの数は1つ以上4つ以下になるように設定します。
また、ダウンスケール安定化ウィンドウを30秒に設定します。
Answer:
タスクの概要
* 自動スケール名前空間に apache-server という名前の HPA を作成します。
* apache-server という名前の既存のデプロイメントをターゲットにします。
* CPUターゲット: 50%
* ポッド範囲: 最小 1、最大 4
* ダウンスケール安定化ウィンドウ: 30秒
ステップバイステップの回答
ステップ1: 正しいホストに接続する
警告画像に示されているように、これは重要です。
ssh cka000050
これをスキップすると、この質問の回答が 0 になる可能性があります。
ステップ2: デプロイメントが存在することを確認する
kubectl デプロイメントの取得 apache-server -n 自動スケール
HPAを作成する前に、必ずこのディレクトリが存在することを確認してください。存在しない場合、HPAは正しくバインドされません。

ステップ3: HPAを作成する
kubectl autoscale コマンドを使用して簡単にセットアップし、その後パッチを適用して安定化ウィンドウを追加します (kubectl autoscale には安定化ウィンドウが含まれていないため)。
kubectl 自動スケール デプロイメント apache-server \
--namespace 自動スケール \
--CPUパーセント=50 \
--min=1 \
--最大=4
ステップ4: ダウンスケール安定化ウィンドウを追加する
30 秒の安定化ウィンドウを含めるように HPA にパッチを適用する必要があります。
hpa-patch.yaml というパッチ ファイルを作成します。
仕様:
行動：
スケールダウン:
安定化ウィンドウ秒数: 30
パッチを適用します:
バッシュ
コピー編集
kubectl パッチ hpa apache-server \
-n 自動スケール \
--patch "$(cat hpa-patch.yaml)"
ステップ5: 作業を確認する
バッシュ
コピー編集
kubectl で hpa apache-server -n オートスケールを記述します
探す：
* 最小/最大ポッド: 1/4
* 目標CPU使用率: 50%
* 安定化ウィンドウ: 動作 > スケールダウンの下に表示されます
ssh cka000050
kubectl デプロイメントの取得 apache-server -n 自動スケール
kubectl 自動スケール デプロイメント apache-server \
--namespace 自動スケール \
--CPUパーセント=50 \
--min=1 \
--最大=4
安定化ウィンドウを追加するパッチ
cat <<EOF > hpa-patch.yaml
仕様:
行動：

スケールダウン:
安定化ウィンドウ秒数: 30
EOF
kubectl patch hpa apache-server -n autoscale --patch "$(cat hpa-patch.yaml)"

最新問題**: 20**
スコア: 4%

タスク
次のようにポッドをスケジュールします。
* 名前: nginx-kusc00401
* 画像: nginx
* ノードセレクター: ディスク=ssd
Answer:
以下の解決策を参照してください。
説明
解決：
#yaml
APIバージョン: v1
種類: ポッド
メタデータ:
名前: nginx-kusc00401
仕様:
コンテナ:
- 名前: nginx
画像: nginx
イメージプルポリシー: IfNotPresent
ノードセレクター:
ディスク：回転
#

kubectl create -f ノード選択.yaml

最新問題**: 21**
次のタスクを実行します。
hungry-bear に init コンテナを追加します (これはスペックファイル /opt/KUCC00108/pod-spec-KUC C00108.yaml で定義されています)。init コンテナは /workdir/calm.txt という名前の空のファイルを作成する必要があります。/workdir/calm.txt が検出されない場合、ポッドは終了します。スペックファイルが init コンテナの定義で更新されると、ポッドが作成されます。
Answer:
解決


Readme
Web Terminal
THE LINUX FOUNDATION
apiVersion: v1
kind: Pod
metadata:
name: hungry-bear
spec:
volumes:
- name: workdir
emptyDir:
containers:
- name: checker
image: alpine
volumeMounts:
- name: workdir
mountPath: /workdir
initContainers:
- name: create
image: alpine
volumeMounts:
- name: workdir
mountPath: /workdir
:wq
root@node-1:~# vim ds.yaml
iroot@node-1:~# k create -f ds.yaml
daemonset.apps/ds-kusc00201 created
root@node-1:~# k get ds
NAME DESIRED CURRENT READY UP-TO-DATE AVAILABLE NODE SELECTOR AGE
ds-kusc00201 2 2 2 2 2 <none> 4s
root@node-1:~# vim /opt/KUCC00108/pod-spec-KUCC00108.yaml
root@node-1:~# k create -f /opt/KUCC00108/pod-spec-KUCC00108.yaml
pod/hungry-bear created
root@node-1:~#

最新問題: **22**

Kubernetes クラスターがあり、異なるチームが異なる名前空間でアプリケーションを管理しています。特定の名前空間でリソースを管理できるようにしながら、他の名前空間のリソースへのアクセスを制限したいと考えています。

Answer:

以下のステップバイステップの説明付きの解決策を参照してください。

Explanation:

解決策（ステップバイステップ）：

1. チームのサービスアカウントを作成します。

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: team-sa
  namespace: team-namespace
```

2. チームの ClusterRole を作成します。

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: team-clusterrole
rules:
- apiGroups: ["apps"]
  resources: ["deployments", "statefulsets", "daemonsets", "replicasets", "pods", "services"]
  verbs: ["create", "get", "list", "watch", "update", "delete"]
- apiGroups: ["core"]
  resources: ["namespaces"]
  verbs: ["get", "list", "watch"]
```

3. ClusterRole を ServiceAccount にバインドし、特定の名前空間へのアクセスを制限する ClusterRoleBinding を作成します。

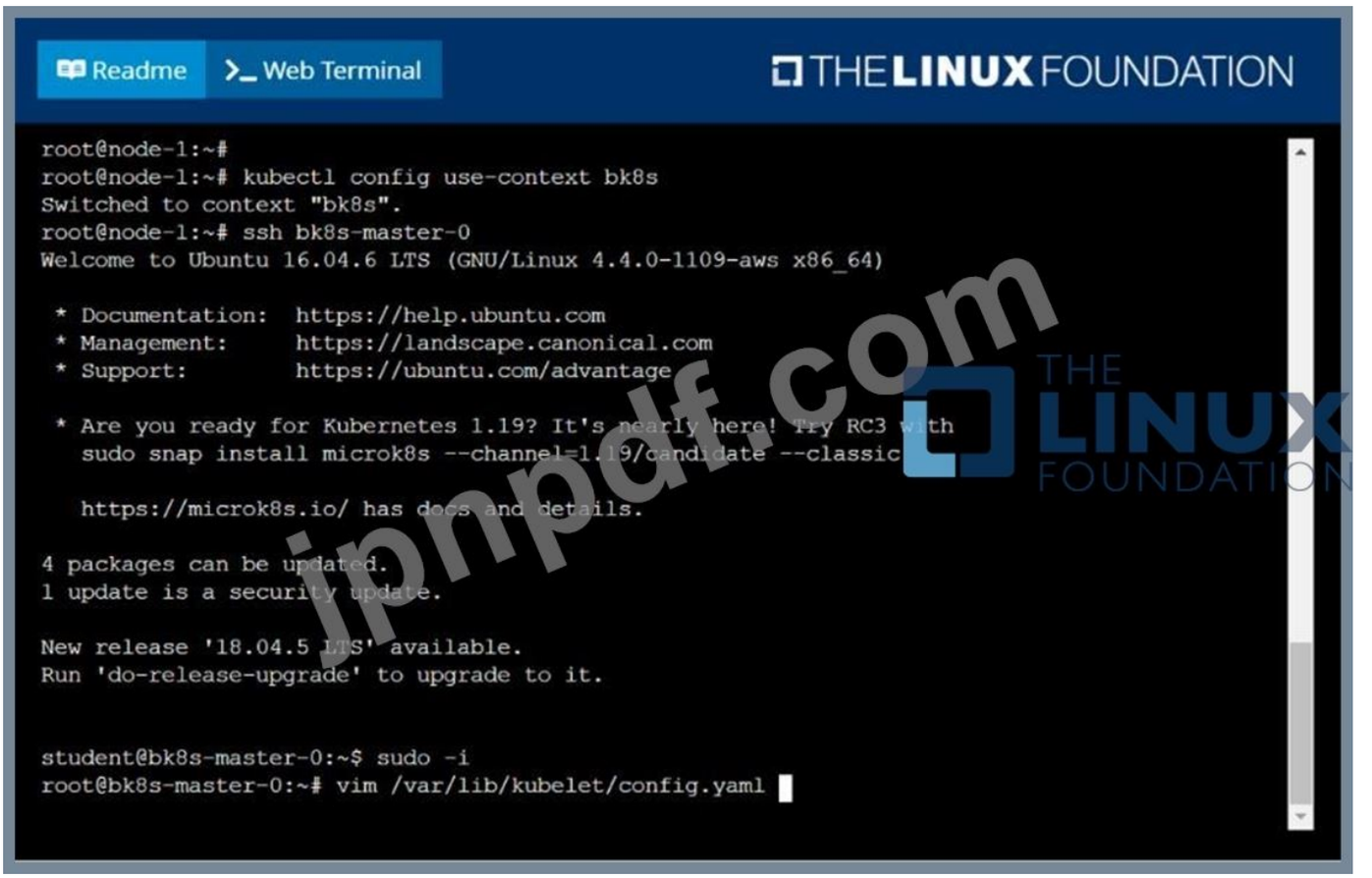
```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: team-clusterrolebinding
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: team-clusterrole
subjects:
- kind: ServiceAccount
  name: team-sa
  namespace: team-namespace
```

4. 「team-sa」、「team-namespace」、「team-clusterrole」を実際の名前に置き換えます。5. 割り当てられた名前空間に ServiceAccount としてデプロイメントを作成し、他の名前空間のリソースにアクセスできないことを確認して、構成をテストします。

最新問題: **23**
「frontend」という名前のポッド ログを一覧表示し、パターン「started」を検索して、ファイル「/opt/error-logs」に書き込みます。以下の解決策を参照してください。
Answer:
Kubectl ログ フロントエンド | grep -i "started" > /opt/error-logs

最新問題: **24**
既存のポッド名のCPUとメモリの要求と制限を設定する
「nginx-prod」。
CPUとメモリの要求をそれぞれ100mと256Miに設定する
CPUとメモリの制限をそれぞれ200MBと512MBに設定する
A. kubectl get を実行後
kubectl で nginx-prod のリソースを設定します --
制限=CPU=200m、メモリ=512Mi --requests=CPU=100m、メモリ=256Mi
//確認する
kubectlトップpo
kubectl describe po nginx-prod
B. kubectl get を実行後
kubectl で nginx-prod のリソースを設定します --
制限=CPU=200m、メモリ=512Mi --requests=CPU=100m、メモリ=256Mi
//確認する
kubectl describe po nginx-prod
Answer: A (メーッセージを残す**)**

最新問題: **25**
部分的に機能している Kubernetes クラスターを前提として、クラスター上の障害の症状を特定します。
障害が発生しているノードとサービスを特定し、障害が発生したサービスを起動してクラスターの健全性を回復するための措置を講じます。変更は必ず永続的に行ってください。
関連するIノードにSSHで接続することができます（
[student@node-1] $ ssh <ノード名
次のコマンドを使用すると、クラスター内の任意のノードで昇格された権限を取得できます。
[学生@ノード名] $ | sudo -i
Answer:
以下の解決策を参照してください。
説明
解決
F:\Work\Data Entry Work\Data Entry\20200827\CKA\23 C.JPG



F:\Work\Data Entry Work\Data Entry\20200827\CKA\23 D.JPG

F:\Work\Data Entry Work\Data Entry\20200827\CKA\23 E.JPG

最新問題: **26**

この項目では、ik8s-master-0 ノードと ik8s-node-0 ノードに SSH 接続し、これらのノード上のすべてのタスクを完了する必要があります。この項目を完了したら、必ずベースノード ホスト名：node-1）に戻ってください。

コンテクスト

小規模な開発チームの管理者として、新しいアプリケーションの実行可能性をテストするために Kubernetes クラスターをセットアップするように依頼されました。

タスク

このタスクを実行するにはkubeadmを使用する必要があります。kubeadmの呼び出しでは、必ず--ignore-preflight-errors=allオプションを使用してください。

* ノード ik8s-master-O をマスターノードとして設定します。.

* ノード ik8s-node-o をクラスターに参加させます。

Answer:

クラスターを初期化するときは、/etc/kubeadm.conf にある kubeadm 構成ファイルを使用する必要があります。

このタスクを完了するには任意の CNI プラグインを使用できますが、お気に入りの CNI プラグインのマニフェスト URL が手元にない場合は、Calico が一般的なオプションの 1 つです: https://docs.projectcalico.org/v3.14/manifests/calico.yaml Docker はすでに両方のノードにインストールされており、必要なツールをインストールできるように apt が構成されています。

最新問題: **27**
イメージバージョン1.16.1でデプロイメントを更新し、イメージを検証してロールアウト履歴を確認します。
Answer:
kubectl set image deploy/webapp nginx=nginx:1.16.1 kubectl describe deploy webapp | grep Image kubectl rollout history deploy webapp

最新問題: **28**
この項目では、ik8s-master-0 ノードと ik8s-node-0 ノードに SSH 接続し、これらのノード上のすべてのタスクを完了する必要があります。この項目を完了したら、必ずベースノード ホスト名：node-1）に戻ってください。
コンテクスト
小規模な開発チームの管理者として、新しいアプリケーションの実行可能性をテストするために Kubernetes クラスターをセットアップするように依頼されました。
タスク
このタスクを実行するにはkubeadmを使用する必要があります。kubeadmの呼び出しには、--ignore-preflight-errors=all オプション。
* ノード ik8s-master-O をマスターノードとして設定します。.
* ノード ik8s-node-o をクラスターに参加させます。
Answer:
以下の解決策を参照してください。
説明
解決
クラスターを初期化するときは、/etc/kubeadm.conf にある kubeadm 構成ファイルを使用する必要があります。
このタスクを完了するには任意の CNI プラグインを使用できますが、お気に入りの CNI プラグインのマニフェスト URL が手元にない場合は、Calico が一般的な選択肢の 1 つです: https://docs.projectcalico.org/v3.14/manifests/calico.yaml Docker はすでに両方のノードにインストールされており、必要なツールをインストールできるように構成されています。

最新問題: **29**
ファイルを作成します:
名前空間開発でサービスを実装するすべてのポッドをリストする /opt/KUCC00302/kucc00302.txt。
ファイルの形式は、1 行につき 1 つのポッド名にする必要があります。
Answer:
以下の解決策を参照してください。
説明
解決


Readme
Web Terminal
THE LINUX FOUNDATION
root@node-1:~#
root@node-1:~# k describe svc baz -n development
Name: baz
Namespace: development
Labels: <none>
Annotations: <none>
Selector: name=foo
Type: ClusterIP
IP: 10.104.252.175
Port: <unset> 80/TCP
TargetPort: 9376/TCP
Endpoints: 10.244.1.5:9376,10.244.2.3:9376,10.244.2.6:9376
Session Affinity: None
Events: <none>
root@node-1:~# k get po -l name=foo -n development
NAME READY STATUS RESTARTS AGE
pod-kucc00302-847878 1/1 Running 0 6h35m
pod-kucc00302-983457 1/1 Running 0 6h35m
pod-kucc00302-985953 1/1 Running 0 6h35m
root@node-1:~# k get po -l name=foo -n development -o NAME
pod/pod-kucc00302-847878
pod/pod-kucc00302-983457
pod/pod-kucc00302-985953
root@node-1:~# k get po -l name=foo -n development -o NAME > /opt/KUCC00302/kucc00302.txt
root@node-1:~# vim /opt/KUCC00302/kucc00302.txt


Readme
Web Terminal
THE LINUX FOUNDATION
pod-kucc00302-847878
pod-kucc00302-983457
pod-kucc00302-985953
:wq

最新問題**: 30**
5つのレプリカを持つイメージnginxを含むwebappというデプロイメントを作成し、webapp.yaml という名前のファイルを/tmpディレクトリに配置します。
A. //dry run コマンドを使用してファイルを作成する
kubectl create deploy --image=nginx --dry-run -o yaml >
/tmp/webapp.yaml
// 次に、webapp.yaml ファイルを編集し、replicas=5 を更新します。
APIバージョン: apps/v1
種類: デプロイメント
メタデータ:
ラベル:
アプリ: ウェブアプリ
名前: ウェブアプリ
仕様:
レプリカ: 5
セレクタ：
一致ラベル:
アプリ: ウェブアプリ
テンプレート：
メタデータ:
ラベル:

アプリ: ウェブアプリ
仕様:
コンテナ:
- 画像: nginx
名前: nginx
注: kubernetes.ioサイトで「deployment」を検索すると、
ページ
https://kubernetes.io/docs/concepts/workloads/controllers/deplo
注釈/
// デプロイメントの検証
kubectl get deploy webapp --show-labels
// デプロイメントWebアプリケーションのYAMLファイルを出力します
kubectl get deploy webapp -o yaml
B. //dry run コマンドを使用してファイルを作成する
kubectl create deploy --image=nginx --dry-run -o yaml >
/tmp/webapp.yaml
// 次に、webapp.yaml ファイルを編集し、replicas=5 を更新します。
APIバージョン: apps/v1
種類: デプロイメント
メタデータ:
ラベル:
アプリ: ウェブアプリ
名前: ウェブアプリ
仕様:
レプリカ: 5
セレクタ :
一致ラベル:
アプリ: ウェブアプリ
テンプレート :
メタデータ:
ラベル:
注: kubernetes.ioサイトで「deployment」を検索すると、
ページ
https://kubernetes.io/docs/concepts/workloads/controllers/deplo
注釈/
// デプロイメントの検証
kubectl get deploy webapp --show-labels
// デプロイメントWebアプリケーションのYAMLファイルを出力します
kubectl get deploy webapp -o yaml
Answer: A (メーッセージを残す**)**

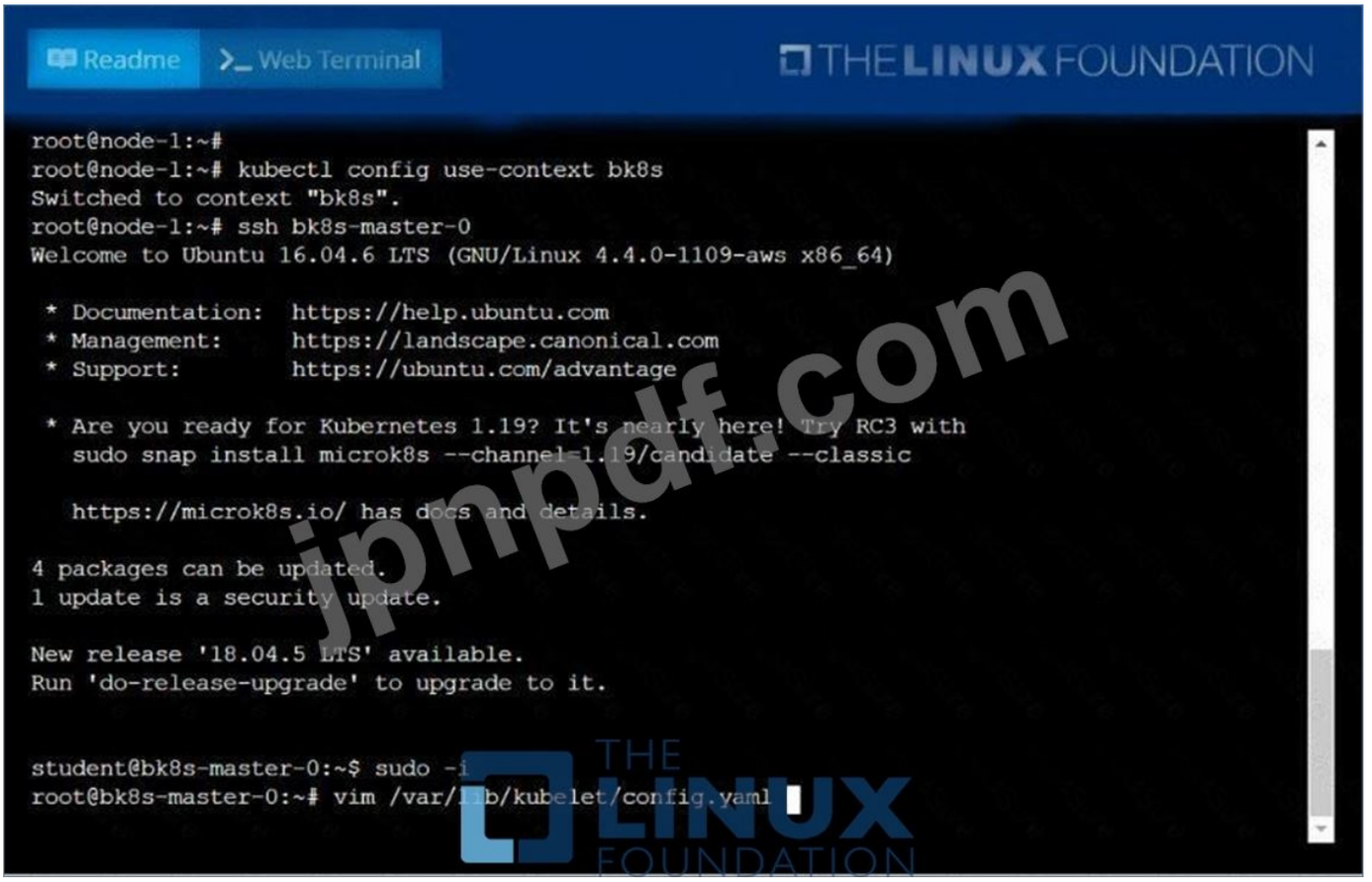
最新問題: **31**
「config.txt」というファイルを作成し、key1=value1という2つの値を設定します。
そして、key2=value2です。次に、「keyvalcfgmap」という名前のconfigmapを作成し、「config.txt」ファイルからデータを読み取り、configmapが正しく作成されたことを確認します。
A. cat >> config.txt << EOF
キー1 = 値1
キー2 = 値2
EOF
kubectl で cm keyvalcfgmap を作成します --from-file=config.txt
//確認する
kubectl get cm keyvalcfgmap -o yaml
B. cat >> config.txt << EOF
キー1 = 値1
キー2 = 値2
EOF
猫の設定.txt
// "config.txt" ファイルから configmap を作成する
kubectl で cm keyvalcfgmap を作成します --from-file=config.txt
//確認する
kubectl get cm keyvalcfgmap -o yaml
Answer: (解答を表示する**)**

最新問題: **32**
jsonpath 式を使用して名前と名前空間を表示するすべてのポッドのリストを取得します。
Answer:
kubectl get pods -o=jsonpath="{.items[*]['metadata.name' , 'metadata.namespace']}"

最新問題: **33**
基本的な WildFire サービスの一部としてサポートされているファイル タイプ アップロードは何ですか?
A. オン

B. VBS
C. エルフ
D. バット
Answer: (解答を表示する**)**

最新問題**: 34**
部分的に機能している Kubernetes クラスターを前提として、クラスター上の障害の症状を特定します。
障害が発生しているノードとサービスを特定し、障害が発生したサービスを起動してクラスターの健全性を回復するための措置を講じます。変更は必ず永続的に行ってください。
次のコマンドを使用して、関連する I ノード (bk8s-master-0 または bk8s-node-0) に ssh 接続できます。
[student@node-1] $ ssh <ノード名>
次のコマンドを使用すると、クラスター内の任意のノードで昇格された権限を取得できます。
[学生@ノード名] $ | sudo -i
Answer:
以下の解決策を参照してください。
説明
解決

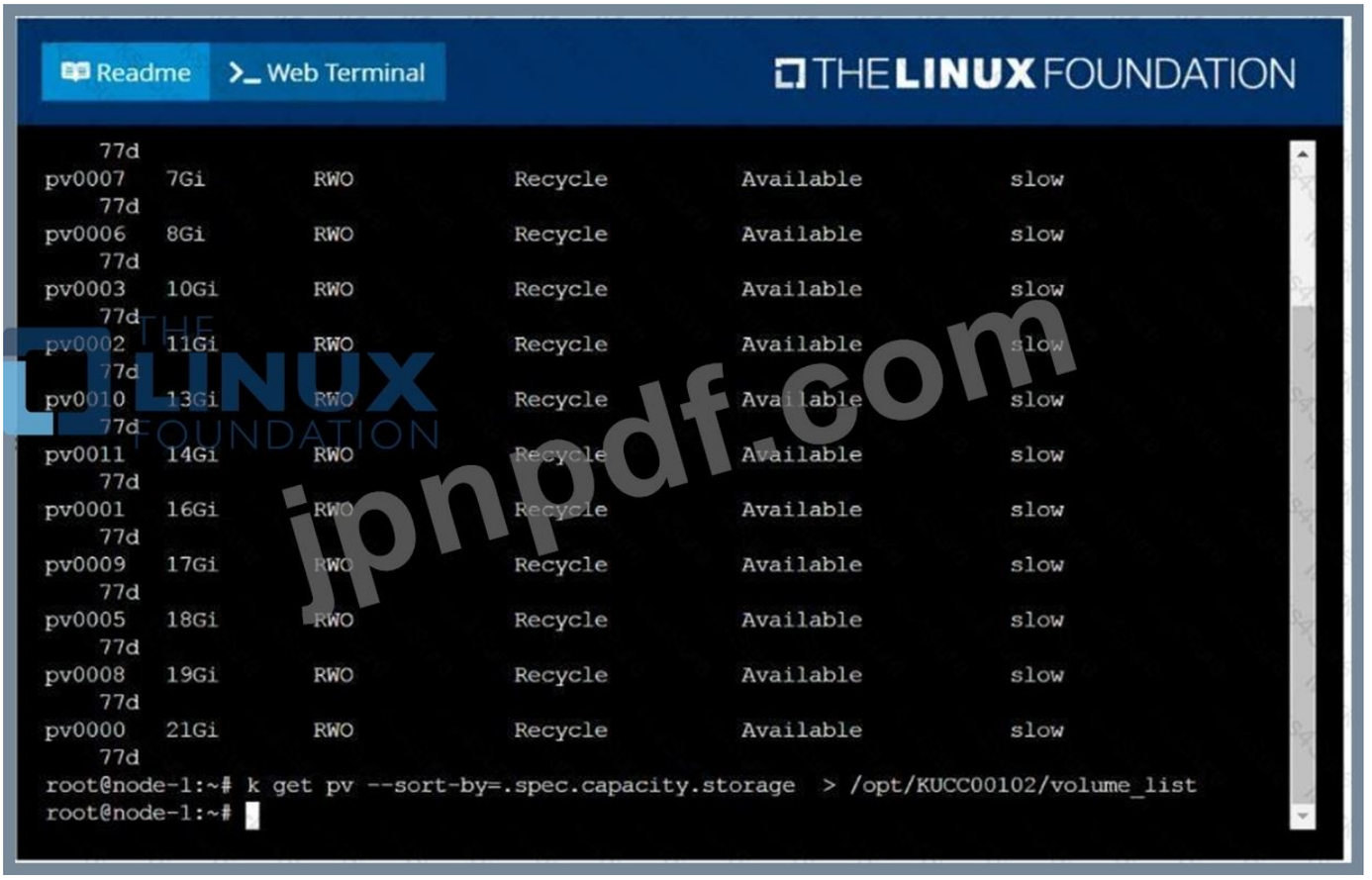


最新問題: **35**

すべての入力トラフィックを拒否するNetworkPolicyを作成する

A. APIバージョン: networking.k8s.io/v1
種類: ネットワークポリシー
メタデータ:
名前: デフォルト拒否
仕様:
ポッドセレクター: {}
ポリシータイプ:
- イングレス
B. apiバージョン: networking.k8s.io/v1
種類: ネットワークポリシー
メタデータ:
名前: デフォルト拒否
仕様:
ポッドセレクター: ()
ポリシータイプ:
- イングレス
Answer: A (メーッセージを残す**)**

最新問題**: 36**
すべての永続ボリュームを容量順に一覧表示し、完全な kubectl 出力
を /opt/KUCC00102/volume_list に保存します。
出力をソートするには kubectl 独自の機能を使用し、それ以上操作しないでください。
Answer:



最新問題: **37**
更新したポッドのイメージバージョンを 1.17.1 に戻し、変更を確認します。
Answer:
kubectl set image pod/nginx nginx=nginx:1.17.1 kubectl describe po nginx kubectl get po nginx -w
監視する

最新問題: **38**
名前と名前空間を JSON パス式で表示するすべてのポッドを一覧表示します。以下の解決策を参照してください。
Answer:
kubectl get pods -o=jsonpath="{.items[*]['metadata.name',
'metadata.namespace']}"

最新問題: **39**
Kubernetes。マイクロサービスは共有データベースを介して相互に通信します。データベース内の永続データを管理し、すべてのマイクロサービスの可用性とスケーラビリティを確保する戦略をどのように実装するかを説明してください。
Answer:
以下のステップバイステップの説明付きの解決策を参照してください。
Explanation:
解決策 ステップバイステップ） :

1. 高可用性データベースを使用する:
- Galeraを使用したMySQLやPatroniを使用したPostgreSQLなど、高可用性をサポートするデータベースシステムを選択してください。これらのデータベースシステムは、複数のノード間でデータを複製できるため、フォールトトレランスとスケーラビリティが実現します。
2. データベースをStatefulSetとしてデプロイする:
- データベースデプロイメント用のStatefulSetを作成し、各ポッドに一意の名前とボリュームクレームを割り当てます。これにより、ポッドが再起動または削除された場合でも、データベースのデータが保持されます。
3. 永続ボリュームとクレームを実装する:
- 各データベース ノードに対して PersistentVolumeClaims (PVC) を定義し、必要なパフォーマンスと回復力を提供するストレージ クラスを要求します。
- これらの PVC をサポートするために、対応する PersistentVolumes (PV) を作成し、十分な容量と適切なアクセス モードを確保します。
4. データベースにアクセスするためのマイクロサービス ポッドを構成する:
- StatefulSet のサービス名または専用のデータベース サービスを使用してデータベースにアクセスするように各マイクロサービス ポッドを構成します。
5. サービスメッシュを活用する:
- マイクロサービスとデータベース間の通信を管理するために、Istio のようなサービスメッシュの導入を検討してください。サービスメッシュは、負荷分散、サービス検出、セキュリティなどの機能を提供し、通信管理を簡素化します。
6. 監視とアラートを実装する:
- データベースとマイクロサービスの健全性とパフォーマンスを監視し、問題を迅速に検出して解決します。重大なイベントや障害を通知するアラートを設定します。
7. 必要に応じてデータベースを拡張する:
- 水平ポッドオートスケーリング（HPA）を使用して、データベースのデプロイメントを負荷に基づいて自動的にスケーリングします。これにより、データベースが増加するトラフィックに対応できるようになります。

最新問題**: 40**

「frontend」という名前のポッドログを一覧表示し、「started」というパターンを検索して、ファイル「/opt/error-logs」に書き込みます。

Answer:

以下の解決策を参照してください。

説明

Kubectl ログ フロントエンド | grep -i "started" > /opt/error-logs

最新問題**: 41**

チームはKubernetes上に重要なアプリケーションをデプロイしており、その可用性とパフォーマンスを確保する必要があります。アプリケーションにロードバランサを実装し、トラフィックを

複数のポッドに分散することを検討しています。Kubernetesで利用可能なロードバランサの種類と、クラウドプロバイダーのロードバランササービスを使用して外部ロードバランサを実装する方法を説明してください。

Answer:

以下のステップバイステップの説明付きの解決策を参照してください。

Explanation:

解決策（ステップバイステップ）：

1. Kubernetes のロードバランサーの種類:

- NodePort: 各ノードの IP アドレスと特定のポート上でサービスを公開するシンプルなロード バランサー。
- LoadBalancer: クラウド プロバイダーのロード バランサーのパブリック IP アドレスでサービスを公開します。
- Ingress: サービスへのトラフィックのより柔軟なルーティングと構成を可能にする高レベルの抽象化。

2. クラウドプロバイダーを使用した外部ロードバランサーの実装:

- Kubernetes サービスを作成します。
- 特定のポート上でアプリケーションを公開する Kubernetes サービスを定義します。
- サービス タイプを「LoadBalancer」に設定します。

```
apiVersion: v1
kind: Service
metadata:
  name: myapp-service
  namespace: myapp-namespace
spec:
  type: LoadBalancer
  ports:
  - port: 80
    targetPort: 8080
  selector:
    app: myapp
```


- クラウド プロバイダーのロードバランサーを構成する: - クラウド プロバイダー (AWS Elastic Load Balancer、Google Cloud Load Balancing、Azure Load Balancer など) のロードバランサー管理コンソールにアクセスします。- 新しいロードバランサーを作成し、目的のポート (ポート 80 など) をリッスンするよう構成します。- トラフィックを Kubernetes サービスに分散するようにロードバランサーを構成します。クラウド プロバイダーの設定によっては、Kubernetes サービスの IP アドレスまたはホスト名の指定が必要になる場合があります。- ロードバランサーが正常なポッドにのみトラフィックをルーティングするようにヘルスチェック設定を構成します。- ロードバランサーの構成を確認する: - クラウド プロバイダーのロードバランサーを構成したら、ロードバランサーのパブリック IP アドレスにアクセスし、アプリケーションが期待どおりに応答することを確認することで、ロードバランサーが正しく動作していることを確認します。- 'kubectl

describe service myapp-service' を使用して、ロードバランサーのステータスと外部 IP アドレスを確認することもできます。,

最新問題: **42**
すべての名前空間内のすべてのポッドのリストを取得し、ファイル「/opt/pods-list.yaml」に書き込みます。
Answer:
kubectl get po -all-namespaces > /opt/pods-list.yaml

最新問題: **43**
複数の名前空間を持つKubernetesクラスターを管理しています。「production」名前空間へのアクセスを制限し、承認されたユーザーのみがその名前空間内のリソースにアクセスできるようにする必要があります。「developers」グループのユーザーが「production」名前空間内のポッドとデプロイメントにアクセスできるようにするロールとロールバインディングを作成してください。
Answer:
以下のステップバイステップの説明付きの解決策を参照してください。
Explanation:
解決策（ステップバイステップ）：
ステップ1: ロールを作成する
次の権限を持つ「production-access」という名前のロールを作成します。

```
kubectl create role production-access --namespace=production --verb=get,list,watch,create,update,patch,delete --resource=pods,deployments
```

ステップ 2: RoleBinding を作成する 「production-access」ロールを「developers」グループにバインドする「production-developers」という名前の RoleBinding を作成します。

```
kubectl create rolebinding production-developers --namespace=production --role=production-access --group=developers
```

ステップ 3: 検証 ロールとロールバインディングが正しく作成されたことを確認します: kubectl get role - -namespace=production kubectl get rolebinding - -namespace=production

最新問題: **44**
pod foo のログを監視し、次の操作を実行します。
対応するログ行を抽出
ウェブサイトにアクセスできない
/opt/KULM00201/foo に書き込む

Answer:

以下の解決策を参照してください。

説明

解決

F:\Work\Data Entry Work\Data Entry\20200827\CKA\1 B.JPG

F:\Work\Data Entry Work\Data Entry\20200827\CKA\1 C.JPG

最新問題**: 45**
スコア: 4%

タスク
次のようにポッドをスケジュールします。
* 名前: nginx-kusc00401
* 画像: nginx
* ノードセレクター: ディスク=ssd
Answer:
解決：

#yaml
APIバージョン: v1
種類: ポッド
メタデータ:
名前: nginx-kusc00401
仕様:
コンテナ:
- 名前: nginx
画像: nginx
イメージプルポリシー: IfNotPresent
ノードセレクター:
ディスク：回転
#
kubectl create -f ノード選択.yaml

最新問題**: 46**
3つのレプリカを持つMySQLデータベースを稼働させている「mysql-cluster」というStatefulSetがあります。既存のデータベース操作を中断することなく、新しいレプリカをクラスタに追加したいと考えています。データの整合性を確保し、ダウンタイムを最小限に抑えながら、これを実現するにはどうすればよいでしょうか？
Answer:
以下のステップバイステップの説明付きの解決策を参照してください。
Explanation:
解決策（ステップバイステップ）：
1. StatefulSet をスケールアップする:
- StatefulSet定義の「レプリカ」値を3から4に増やします。「kubectl apply -f mysql-cluster.yaml」を使用して変更を適用します。

```
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: mysql-cluster
spec:
  replicas: 4
  # ... other StatefulSet configuration ...
```

2. 新しいポッドが作成されるまで待機します。- 「kubectl get pods -l app=mysql-cluster」を使用して、ポッドの作成プロセスを監視します。新しいポッドが作成され、準備完了状態になるまで待機します。3. 新しいポッドをクラスタに参加させます。- 新しいポッドのシェルで、以下のコマンドを実行して既存のMySQLクラスタに参加させます。bash mysql -h -u -p -e "CHANGE MASTER TO MASTER PASSWORD=", MASTER DELAY=O" - は既存のMySQLレプリカのIPアドレスに置き換えます。- , および '3306' は、MySQLの設定に適した値に置き換えます。4. 新しいレプリカが

同期されていることを確認します。- 新しいレプリカで「SHOW SLAVE STATUS」コマンドを使用して、既存のクラスタからデータが正常に複製されていることを確認します。「Slave 10 Running」と「Slave SQL Running」のステータスが両方とも「Yes」になっていることを確認します。5. 新しいレプリカを昇格させます。- StatefulSet定義を更新して新しいポッドのホスト名を追加することで、新しいレプリカをクラスタの完全なメンバーに昇格させます。これには通常、StatefulSetの「VolumeClaimTemplates」セクションを参照してください。6. クラスターの健全性をテストします。- データベースに対して一連の読み取りおよび書き込み操作を実行し、新しいレプリカが完全に統合され、正常に応答していることを確認します。7. 古いポッドを削除します。- ポッドインデックスが最も低い古いポッドを削除できます。これにより、古いボリュームの自動クリーンアップがトリガーされ、正常で同期されたレプリカのみが残ります。これらの手順に従うことで、ダウンタイムを最小限に抑え、データの一貫性を維持しながら、MySQLクラスターに新しいレプリカを追加できます。

最新問題: **47**
以前ラボで使用されていたPalo Alto Networksファイアウォールの初期設定を読み込むため、Windowsワークステーションを使用してブートストラップUSBフラッシュドライブを準備しました。USBフラッシュドライブはFAT32ファイルシステムでフォーマットされており、初期設定はinit-cfg.txtというファイルに保存されています。ファイアウォールは現在PAN-OS 10.0で実行されており、ラボ設定を使用しています。USBフラッシュドライブ内のinit-cgf.txtの内容は次のとおりです。
タイプ=DHCPクライアント
IPアドレス
デフォルトゲートウェイ
ネットマスク
IPv6アドレス
IPv6デフォルトゲートウェイ
ホスト名=Ca-FW-DC1
パノラマサーバー=10.5.107.20
パノラマサーバー2=10.5.107.21
tplname=FINANCE_TG4
dgname=財務dg
dns-プライマリ=10.5.6.6

dns-セカンダリ=10.5.6.7
オペレーションコマンドモード - マルチvsys.ジャンボフレーム
dhcp-send-hostname=yes
dhcp-send-client-id=yes
DHCP受け入れサーバホスト名=はい
DHCP受け入れサーバードメイン=はい
USBフラッシュドライブをファイアウォールのUSBポートに挿入し、コマンド「request restart system」を使用してファイアウォールを再起動しました。再起動後、ファイアウォールはブートストラッププロセスを開始できません。この失敗の原因は次のとおりです。
A. ブートストラップXMLファイルは必須ファイルですが、見つかりません
B. PAN-OS のバージョンは最低でも 9.1 x である必要がありますが、ファイアウォールは 10.0x を実行しています
C. ファイアウォールは工場出荷時のデフォルト状態になっているか、ブートストラップのためにすべてのプライベートデータが削除されている必要があります。
D. ホスト名は必須パラメータですが、init-cfg.txt にありません。
E. USBはexi3ファイルシステムでフォーマットされている必要があります。FAT32は
Answer: E (メーッセージを残す**)**

最新問題**: 48**
Kubernetes にサービスをデプロイし、別の名前空間で実行されているデータベースサービスにアクセスしようとしています。名前空間をまたいでこれらのサービス間の通信を許可するには、NetworkPolicy をどのように設定すればよいでしょうか？
Answer:
以下のステップバイステップの説明付きの解決策を参照してください。
Explanation:
解決策（ステップバイステップ）：
1. サービスの名前空間にNetworkPolicyを作成します。
- データベースにアクセスする必要があるサービスの名前空間に NetworkPolicy を作成します。
- コード:

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: allow-access-to-database
  namespace: service-namespace # Replace with the namespace of the service
spec:
  podSelector: {}
  egress:
  - to:
    - namespaceSelector:
        matchLabels:
          database: true # Define the label of the database namespace
  policyTypes:
  - Egress
```


2. データベース名前空間に正しいラベルが付いていることを確認します。- データベース サービスが実行されている名前空間に「database: true」というラベルが付いていることを確認します。3. NetworkPolicy を適用します。- 「kubectl apply -f networkpolicy.yaml」を使用して NetworkPolicy を適用します。

最新問題**: 49**
スコア: 7%

タスク
次のように新しい nginx Ingress リソースを作成します。
* 名前: ピン
* 名前空間: ing-internal
* サービスポート 5678 を使用してパス /hi 上のサービス hi を公開する

Answer:
解決：

ingress.yaml
#
APIバージョン: networking.k8s.io/v1
種類: イングレス
メタデータ:
名前: ピン
名前空間: ing-internal
仕様:
ルール:
- http:
パス:
- パス: /hi
パスタイプ: プレフィックス
バックエンド:
サービス：
名前: こんにちは
ポート：
番号: 5678
#
kubectl create -f ingress.yaml

最新問題**: 50**
複数の名前空間を持つKubernetesクラスターがあります。特定のユーザー 「developer」が 「dev」名前空間内のポッドのデプロイと管理のみを許可し、他のリソースへのアクセスを制限するようにRBACを設定する必要があります。
Answer:
以下のステップバイステップの説明付きの解決策を参照してください。
Explanation:
次の YAML ファイルを作成します。
1. Role.yaml:

```yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: pod-manager-dev
  namespace: dev
rules:
  - apiGroups: ["apps"]
    resources: ["deployments", "pods"]
    verbs: ["create", "delete", "get", "list", "patch", "update", "watch"]
  - apiGroups: ["extensions"]
    resources: ["ingresses"]
    verbs: ["get", "list", "watch"]
```

2. RoleBinding.yaml:

```yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: pod-manager-dev-binding
  namespace: dev
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: Role
  name: pod-manager-dev
subjects:
  - kind: User
    name: developer
    apiGroup: rbac.authorization.k8s.io
```

解決方法 (手順): 1. ロールの作成: 'kubectl apply -f Role.yamr' を使用して 'Role.yamP' ファイルを適用します。これにより、'dev' 名前空間内でロール 'pod-manager-dev' に付与される権限が定義されます。2. ロールバインディングの作成: 'kubectl apply -f RoleBinding.yaml' を使用して 'RoleBinding.yaml' ファイルを適用します。これにより、'pod-manager-dev' ロールがユーザー 'developer' にバインドされ、ロールで定義されたリソースにアクセスできるようになります。3. アクセスの確認: ユーザー 'developer' として、'dev' 名前空間内でポッドのデプロイやデプロイの管理を試みます。ユーザーに必要な権限があることを確認します。4. 制限のテスト: 'dev' 名前空間外のリソースへのアクセスや、ロールで定義されていないアクション (例: 'dev' 名前空間でのサービスの作成) の試行を試します。ユーザーのアクセスが拒否されていることを確認します。

最新問題**: 51**
ek8s-node-1 という名前のノードを使用不可に設定し、そのノードで実行されているすべてのポッドを再スケジュールします。
Answer:
解決

最新問題: **52**

「busyboxpod-{1,2,3}」の各コンテナのログを確認します。

A. kubectl ログ busybox -c busybox-container-1
kubectl ログ busybox -c busybox-container-3
kubectl ログ busybox -c busybox-container-3

B. kubectl ログ busybox -c busybox-container-1
kubectl ログ busybox -c busybox-container-2
kubectl ログ busybox -c busybox-container-3

Answer: B (メーッセージを残す**)**

最新問題: **53**

RBAC を実装して Kubernetes クラスターを保護するという課題があります。クラスターには「dev」と「prod」という 2 つの名前空間があります。「dev」名前空間のユーザーにはデプロイメントの作成、削除、一覧表示を許可し、「prod」名前空間のデプロイメントには読み取り専用アクセスのみを許可するロールを作成する必要があります。さらに、これらのユーザーには両方の名前空間内で ConfigMap を作成および管理する権限も付与する必要があります。

このアクセス制御ポリシーを実装するために必要な RBAC リソース (Role、RoleBinding) を作成します。

Answer:

以下のステップバイステップの説明付きの解決策を参照してください。

Explanation:
解決策（ステップバイステップ）：
ステップ 1: 「dev」名前空間のロールを作成します。

```
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: dev-deployment-role
  namespace: dev
rules:
- apiGroups: ["apps"]
  resources: ["deployments"]
  verbs: ["create", "delete", "list", "get"]
- apiGroups: ["core"]
  resources: ["configmaps"]
  verbs: ["create", "delete", "list", "get", "update", "patch"]
```


ステップ 2: 'prod' 名前空間のロールを作成します。

```
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: prod-deployment-role
  namespace: prod
rules:
- apiGroups: ["apps"]
  resources: ["deployments"]
  verbs: ["get", "list"]
```


ステップ 3: 'dev' 名前空間に RoleBinding を作成します。

```yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: dev-deployment-binding
  namespace: dev
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: Role
  name: dev-deployment-role
subjects:
- kind: User
  name: dev-user
  apiGroup: rbac.authorization.k8s.io
```

ステップ 4: 'prod' 名前空間に RoleBinding を作成します。

```yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: prod-deployment-binding
  namespace: prod
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: Role
  name: prod-deployment-role
subjects:
- kind: User
  name: dev-user
  apiGroup: rbac.authorization.k8s.io
```

ステップ 5: YAML ファイルをクラスターに適用します。

```
kubectl apply -f dev-deployment-role.yaml
kubectl apply -f prod-deployment-role.yaml
kubectl apply -f dev-deployment-binding.yaml
kubectl apply -f prod-deployment-binding.yaml
```

これで、「dev-user」は「dev」名前空間内のデプロイメントを作成、削除、一覧表示できるようになりました。「prod」名前空間内のデプロイメントは表示のみ可能です。また、両方の名前空間でConfigMapを作成および管理できます。

最新問題**: 54**

「default」名前空間を持つKubernetesクラスターがあります。この名前空間にユーザーがポッドを作成できないようにしたいと考えています。

この制限を実現するには、ClusterRole と ClusterRoleBinding を作成します。

Answer:

以下のステップバイステップの説明付きの解決策を参照してください。

Explanation:

解決策（ステップバイステップ）：

ステップ 1: 「default」名前空間でのポッド作成を拒否する ClusterRole を作成します。

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
name: deny-pod-creation-default
rules:
- apiGroups: ["apps"]
resources: ["pods"]
verbs: ["create"]
resourceNames: ["default"]
namespaceSelector:
matchLabels:
kubernetes.io/metadata.name: default
```

ステップ 2: ClusterRole をすべてのユーザーに適用するための ClusterRoleBinding を作成します。

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
name: deny-pod-creation-default-binding
roleRef:
apiGroup: rbac.authorization.k8s.io
kind: ClusterRole
name: deny-pod-creation-default
subjects:
- kind: User
apiGroup: rbac.authorization.k8s.io
name: ""
```


「default」名前空間を明示的にターゲットとし、ポッドの「create」権限を拒否するClusterRoleを作成します。このClusterRoleは、「resourceNames」と「namespaceSelector」を使用して、「default」名前空間を正確にターゲットとします。また、クラスター内のすべてのユーザーにClusterRoleを適用するClusterRoleBindingを作成します。設定の適用：「kubectl apply -f [filename].yaml」を使用して、ClusterRoleおよびClusterRoleBindingのYAMLファイルを適用します。これにより、誰も「default」名前空間にポッドを作成できなくなります。この設定は、セキュリティポリシーを実装し、機密性の高い名前空間での意図しないリソース作成を防ぐための強力な方法です。

最新問題**: 55**
次のようにポッドを作成します。
名前: 非永続的Redis
コンテナイメージ: redis
名前がcache-controlのボリューム
マウントパス: /data/redis
ポッドはステージング名前空間で起動する必要があり、ボリュームは永続的であってはなりません。
Answer:
解決


Readme
Web Terminal
THE LINUX FOUNDATION
root@node-1:~#
root@node-1:~#
root@node-1:~# vim volume.yaml


Readme
Web Terminal
THE LINUX FOUNDATION
apiVersion: v1
kind: Pod
metadata:
name: non-persistent-redis
namespace: staging
spec:
containers:
- name: redis
image: redis
volumeMounts:
- name: cache-control
/data/redis
name: cache-control
emptyDir: {}
:w

最新問題**: 56**
環境変数としてユーザー名を読み取るnginxポッドを作成する
A. // ymlファイルを作成する
kubectl run nginx --image=nginx --restart=Never --dry-run -o
yaml > nginx.yml
// 以下にenvセクションを追加して作成します
APIバージョン: v1
種類: ポッド
メタデータ:
ラベル:
実行: nginx
名前: nginx
仕様:
コンテナ:
- 画像: nginx
名前: nginx
環境:
- 名前: USER_NAME
再起動ポリシー: なし
kubectl create -f nginx.yml
//確認する

kubectl exec -it nginx - env
B. // ymlファイルを作成する
kubectl run nginx --image=nginx --restart=Never --dry-run -o
yaml > nginx.yml
// 以下にenvセクションを追加して作成します
APIバージョン: v1
種類: ポッド
メタデータ:
ラベル:
実行: nginx
名前: nginx
仕様:
コンテナ:
- 画像: nginx
名前: nginx
環境:
- 名前: USER_NAME
値の開始:
シークレットキーリファレンス:
名前: 私の秘密
キー: ユーザー名
再起動ポリシー: なし
kubectl create -f nginx.yml
//確認する
kubectl exec -it nginx - env
Answer: B (メーッセージを残す**)**

最新問題**: 57**
https://127.0.0.1:2379 で実行されている etcd インスタンスのスナップショットを作成し、ファイルパスに保存します。
/srv/data/etcd-snapshot.db です。
etcdctl を使用してサーバーに接続するために、次の TLS 証明書/キーが提供されます。
* CA証明書: /opt/KUCM00302/ca.crt
* クライアント証明書: /opt/KUCM00302/etcd-client.crt
* クライアントキー: Topt/KUCM00302/etcd-client.key
Answer:

最新問題: **58**

ユーザーがポッドを作成および削除できるようにする新しいロールを作成する必要がありますが、

「production」名前空間。このロールを「kubectl」コマンドでどのように定義しますか？

Answer:

以下のステップバイステップの説明付きの解決策を参照してください。

Explanation:

解決策（ステップバイステップ）：

1. ロール YAML ファイルを作成します。

```
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: pod-admin
  namespace: production
rules:
- apiGroups: ["apps"]
  resources: ["deployments"]
  verbs: ["get", "list", "watch"]
- apiGroups: ["apps"]
  resources: ["deployments/finalizers"]
  verbs: ["update"]
- apiGroups: ["apps"]
  resources: ["statefulsets"]
  verbs: ["get", "list", "watch"]
- apiGroups: ["apps"]
  resources: ["statefulsets/finalizers"]
  verbs: ["update"]
- apiGroups: ["extensions"]
  resources: ["daemonsets"]
  verbs: ["get", "list", "watch"]
- apiGroups: ["extensions"]
  resources: ["daemonsets/finalizers"]
  verbs: ["update"]
- apiGroups: ["extensions"]
  resources: ["ingresses"]
  verbs: ["get", "list", "watch"]
- apiGroups: ["extensions"]
  resources: ["ingresses/finalizers"]
  verbs: ["update"]
- apiGroups: ["extensions"]
  resources: ["replicasets"]
  verbs: ["get", "list", "watch"]
- apiGroups: ["extensions"]
  resources: ["replicasets/finalizers"]
  verbs: ["update"]
- apiGroups: ["batch"]
  resources: ["jobs"]
  verbs: ["get", "list", "watch"]
- apiGroups: ["batch"]
  resources: ["jobs/finalizers"]
  verbs: ["update"]
- apiGroups: ["batch"]
  resources: ["cronjobs"]
  verbs: ["get", "list", "watch"]
- apiGroups: ["batch"]
  resources: ["cronjobs/finalizers"]
  verbs: ["update"]
- apiGroups: ["", "apps", "extensions", "batch"]
  resources: ["pods"]
  verbs: ["get", "list", "watch", "create", "delete", "update"]
- apiGroups: ["", "apps", "extensions", "batch"]
  resources: ["pods/status"]
  verbs: ["get"]
- apiGroups: ["apps"]
  resources: ["deployments/status"]
  verbs: ["get"]
- apiGroups: ["apps"]
  resources: ["statefulsets/status"]
  verbs: ["get"]
```

```
- apiGroups: ["extensions"]
  resources: ["daemonsets/status"]
  verbs: ["get"]
- apiGroups: ["extensions"]
  resources: ["replicasets/status"]
  verbs: ["get"]
- apiGroups: ["batch"]
  resources: ["jobs/status"]
  verbs: ["get"]
- apiGroups: ["batch"]
  resources: ["cronjobs/status"]
  verbs: ["get"]
- apiGroups: ["extensions"]
  resources: ["ingresses/status"]
  verbs: ["get"]
```

2. クラスターにロールを適用します: kubectl apply -f pod-admin . yaml 3. ロールをユーザーまたはグループに関連付ける RoleBinding を作成します:

```
kubectl create rolebinding pod-admin-binding --role=pod-admin --subjects=user:user1 --namespace=production
```

「Role」リソースは、特定の名前空間に対する一連の権限を定義します。「rules」フィールドは、APIグループ、リソース、および動詞を指定して、ロールに許可されるアクションを定義します。「apiGroups」フィールドには、権限に関連するKubernetes APIグループがリストされます。「resources」フィールドには、ユーザーがアクセスできるKubernetesリソースが指定されます。「verbs」フィールドには、指定されたリソースに対して許可されるアクションがリストされます。「RoleBinding」は、作成された「Role」を特定のユーザーまたはグループに関連付け、指定された権限を付与します。この場合、ロールの名前は「pod-admin」で、スコープは「production」名前空間に設定されています。このロールにより、ユーザーはポッドの作成と削除、および関連リソースに対するその他のアクションを実行できます。この例は、Kubernetesでロールベースアクセス制御（RBAC）を管理する方法を示しています。権限とバインディングは、特定のセキュリティ要件に合わせて調整できます。

最新問題**: 59**

StatefulSetを使用したステートフルアプリケーションとデータベースアプリケーションを実行するDeploymentがあります。ステートフルアプリケーションの書き込みパフォーマンスに影響を与えずに、増加する読み取りトラフィックに対応できるようデータベースをスケールするにはどうすればよいでしょうか？

Answer:

以下のステップバイステップの説明付きの解決策を参照してください。

Explanation:

解決策（ステップバイステップ）：

1. リードレプリカを使用する:

- プライマリ データベースからデータを複製するデータベースの読み取りレプリカを作成します。
- 読み取り専用操作に読み取りレプリカを使用して、読み取り負荷を分散します。

2. StatefulSet を設定します。

- 読み取り専用操作のために読み取りレプリカにアクセスするように StatefulSet を構成します。
- 読み取りレプリカに別のサービスを使用し、それにアクセスするように StatefulSet を構成します。
3. ロードバランサーを実装する:
- ロード バランサを使用して、読み取りトラフィックを読み取りレプリカに送信し、書き込みトラフィックをプライマリ データベースに書き込みます。
- 読み取り要求には特定のポートを使用し、書き込み要求には別のポートを使用するようにロード バランサーを構成します。
4. パフォーマンスを監視する:
- プライマリ データベースと読み取りレプリカの両方のパフォーマンスを監視します。
- プライマリ データベースの書き込みパフォーマンスに影響を与えずに、読み取りレプリカが読み取り負荷を適切に処理していることを確認します。
5. リードレプリカのスケール:
- 必要に応じて、読み取りトラフィックの増加に対応するために読み取りレプリカの数を拡張します。
- 必要に応じて読み取りレプリカを追加し、ロード バランサの構成を調整してトラフィックを均等に分散します。

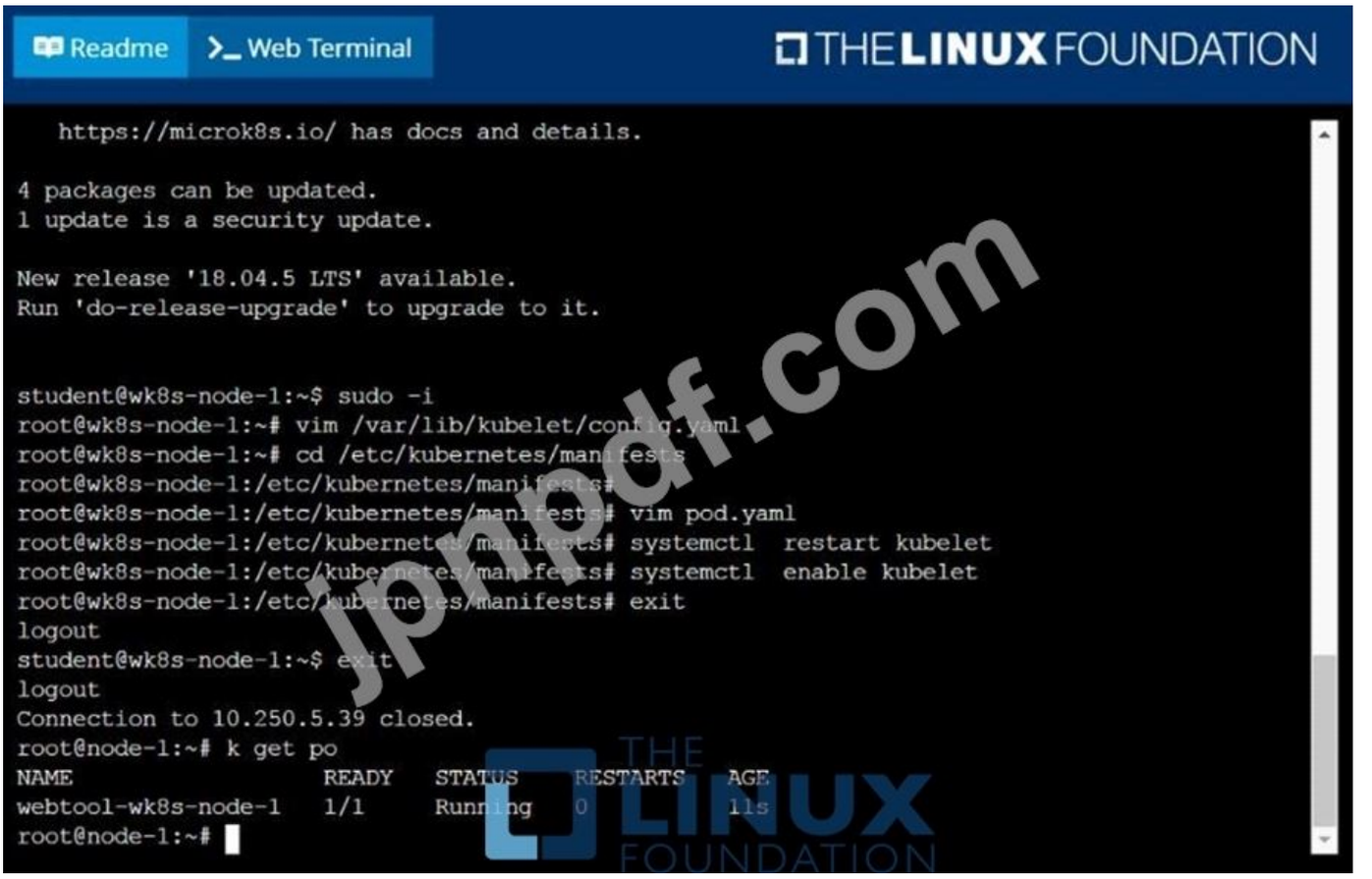
最新問題**: 60**
name=wk8s-node-1 というラベルのノード上で、kubelet systemd 管理サービスを設定し、webtool という名前の Image httpd コンテナを 1 つ含むポッドを自動的に起動します。必要な仕様ファイルは、ノードの /etc/kubernetes/manifests ディレクトリに配置する必要があります。
次のコマンドを使用して適切なノードに SSH 接続できます。
[student@node-1] $ ssh wk8s-node-1
次のコマンドを使用して、ノード上で昇格された権限を取得できます。
[student@wk8s-node-1] $ | sudo -i
Answer:
解決


Readme
Web Terminal
THE LINUX FOUNDATION
root@node-1:~#
root@node-1:~# kubectl config use-context wk8s
Switched to context "wk8s".
root@node-1:~# ssh wk8s-node-1
Welcome to Ubuntu 16.04.6 LTS (GNU/Linux 4.4.0-1109-aws x86_64)
* Documentation: https://help.ubuntu.com
* Management: https://landscape.canonical.com
* Support: https://ubuntu.com/advantage
* Are you ready for Kubernetes 1.19? It's nearly here! Try RC3 with
sudo snap install microk8s --channel=1.19/candidate --classic
https://microk8s.io/ has docs and details.
4 packages can be updated.
1 update is a security update.
New release '18.04.5 LTS' available.
Run 'do-release-upgrade' to upgrade to it.
student@wk8s-node-1:~$ sudo -i
root@wk8s-node-1:~# vim /var/lib/kubelet/config.yaml


Readme
Web Terminal
THE LINUX FOUNDATION
clientCAFile: /etc/kubernetes/pki/ca.crt
authorization:
mode: Webhook
webhook:
cacheAuthorizedTTL: 0s
cacheUnauthorizedTTL: 0s
clusterDNS:
- 10.96.0.10
clusterDomain: cluster.local
cpuManagerReconcilePeriod: 0s
evictionPressureTransitionPeriod: 0s
fileCheckFrequency: 0s
healthzBindAddress: 127.0.0.1
healthzPort: 10248
httpCheckFrequency: 0s
imageMinimumGCAge: 0s
kind: KubeletConfiguration
nodeStatusReportFrequency: 0s
nodeStatusUpdateFrequency: 0s
rotateCertificates: true
runtimeRequestTimeout: 0s
staticPodPath: /etc/kubernetes/manifests
streamingConnectionIdleTimeout: 0s
syncFrequency: 0s
:wq


Readme
Web Terminal
THE LINUX FOUNDATION
root@node-1:~# ssh wk8s-node-1
Welcome to Ubuntu 16.04.6 LTS (GNU/Linux 4.4.0-1109-aws x86_64)
* Documentation: https://help.ubuntu.com
* Management: https://landscape.canonical.com
* Support: https://ubuntu.com/advantage
* Are you ready for Kubernetes 1.19? It's nearly here! Try RC3 with
sudo snap install microk8s --channel=1.19/candidate --classic
https://microk8s.io/ has docs and details.
4 packages can be updated.
1 update is a security update.
New release '18.04.5 LTS' available.
Run 'do-release-upgrade' to upgrade to it.
student@wk8s-node-1:~$ sudo -i
root@wk8s-node-1:~# vim /var/lib/kubelet/config.yaml
root@wk8s-node-1:~# cd /etc/kubernetes/manifests
root@wk8s-node-1:/etc/kubernetes/manifests#
root@wk8s-node-1:/etc/kubernetes/manifests# vim pod.yaml


Readme
Web Terminal
THE LINUX FOUNDATION
apiVersion: v1
kind: Pod
metadata:
name: webtool
spec:
containers:
- name: webtool
image: httpd
:w



最新問題: **61**
組織では社内サービスにプライベートDNSサーバーを使用しており、すべてのKubernetesポッドがこのDNSサーバーに対して名前解決を行う必要があります。すべてのDNSリクエストをこのプライベートサーバーに転送するようにCoreDNSを設定する必要があります。
Answer:
以下のステップバイステップの説明付きの解決策を参照してください。
Explanation:
解決策（ステップバイステップ）：
1. 転送機能を使用してCoreDNSを構成する:
- CoreDNS ConfigMap で、「forward」プラグインを構成して、すべての DNS 要求をプライベート DNS サーバーに転送します。

```
.:53 {
    errors
    health
    ready
    # Forward all requests to the private DNS server
    forward . 192.168.1.10 {
        max_tcp_size 4096
        max_udp_size 4096
    }
    cache 30
    reload 10s
}
```


2. DNS 解決をテストする: - クラスター内のポッドから「nslookup」コマンドを使用して、内部サービスの DNS 解決をテストします。- リクエストはプライベート DNS サーバーに転送され、対応するレコードが返される必要があります。

最新問題: **62**
名前順に並べられたすべてのポッドを一覧表示する
Answer:
以下の解決策を参照してください。
説明
kubect1 ポッドを取得 --sort-by=.metadata.name

最新問題: **63**
3 つのコンテナを含むポッドを作成しますか? (マルチコンテナ)
Answer:
以下の解決策を参照してください。
説明
イメージ=nginx、イメージ=redis、イメージ=consul
nginxコンテナの名前を「nginx-container」にします
Redisコンテナの名前を「redis-container」にします
Consulコンテナの名前を「consul-container」とする
コンテナのポッドマニフェストファイルを作成し、コンテナを追加します
残りの画像のセクション

kubectl マルチコンテナ実行 --generator=run-pod/v1 --image=nginx --
ドライラン -o yaml > マルチコンテナ.yaml
＃ それから
vim マルチコンテナ.yaml
APIバージョン: v1
種類: ポッド
メタデータ:
ラベル:
実行: マルチコンテナ
名前: マルチコンテナ
仕様:
コンテナ:
- 画像: nginx
名前: nginxコンテナ
- 画像: redis
名前: redis-container
- 画像: 領事
名前: consul-container
再起動ポリシー: 常に

最新問題**: 64**
スコア: 7%

タスク
既存の名前空間 echo に、allow-port-from-namespace という新しい NetworkPolicy を作成します。この新しい NetworkPolicy によって、名前空間 my-app の Pod が名前空間 echo の Pod のポート 9000 に接続できるようになります。
さらに、新しい NetworkPolicy が次の点であることを確認します。
* ポート9000をリッスンしないポッドへのアクセスは許可されません
* 名前空間 my-app に含まれない Pod からのアクセスは許可されません
Answer:
以下の解決策を参照してください。
説明
解決：
#ネットワーク.yaml
APIバージョン: networking.k8s.io/v1

種類: ネットワークポリシー
メタデータ:
名前: 名前空間からのポートを許可する
名前空間: 内部
仕様:
ポッドセレクター:
マッチラベル: {
}
ポリシータイプ:
- イングレス
入口:
- から：
- ポッドセレクター: {
}
ポート:
- プロトコル: TCP
ポート: 8080
#spec.podSelector 名前空間ポッド
kubectl create -f ネットワーク.yaml

最新問題**: 65**
スコア: 5%

タスク
ポッド ラベル name=cpu-utilizer から、CPU ワークロードの高いポッドを見つけ、最も多くの CPU を消費しているポッドの名前をファイル /opt/KUTR00401/KUTR00401.txt (既に存在) に書き込みます。
Answer:
解決：
kubectl top -l name=CPUユーザー -A
echo 'ポッド名' >> /opt/KUT00401/KUT00401.txt

最新問題**: 66**
app-data という名前、容量 2Gi、アクセスモード ReadWriteMany の永続ボリュームを作成します。ボリュームタイプは hostPath、場所は /srv/app-data です。

Answer:

解決

永続ボリューム

永続ボリュームは、Kubernetes クラスター内のストレージの一部です。永続ボリュームは、ノードと同様にクラスターレベルのリソースであり、どの名前空間にも属しません。管理者によってプロビジョニングされ、特定のファイルサイズを持ちます。そのため、Kubernetes にアプリをデプロイする開発者は、基盤となるインフラストラクチャを意識する必要はありません。開発者がアプリケーションに一定量の永続ストレージを必要とする場合、システム管理者は、プロビジョニングされた永続ボリュームを簡単に利用できるようにクラスターを構成します。

永続ボリュームの作成

kind: PersistentVolume apiVersion: v1 metadata: name:app-data spec: capacity: # 作成する PV の容量を定義します。storage: 2Gi # 要求するストレージの量 accessModes: # 作成するボリュームの権限を定義します - ReadWriteMany hostPath: path: "/srv/app-data" # ボリュームを作成するパス 課題 アクセス モードが ReadWriteMany、ストレージ クラス名が shared、ストレージ容量が 2Gi、ホスト パスが /srv/app-data の、app-data という名前の永続ボリュームを作成します。

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: app-data
spec:
  capacity:
    storage: 2Gi
  accessModes:
    - ReadWriteMany
  hostPath:
      path: /srv/app-data
  storageClassName: shared
~
~
~
~
~
~
~
~
~
~
~
"app-data.yaml" 12L, 194C
```


2. ファイルを保存し、永続ボリュームを作成します。

```
njerry191@cloudshell:~ (extreme-clone-265411)$ kubectl create -f pv.yaml
persistentvolume/pv created
```

3. 永続ボリュームを表示します。

```
njerry191@cloudshell:~ (extreme-clone-265411)$ kubectl get pv
NAME     CAPACITY   ACCESS MODES   RECLAIM POLICY   STATUS      CLAIM   STORAGECLASS   REASON   AGE
app-data 2Gi        RWX            Retain           Available           shared                  31s
```

永続ボリュームのステータスは「available」です。これは、使用可能であり、まだマウントされていないことを意味します。このステータスは、persistentVolume を persistentVolumeClaim にマウントすると変更されます。

永続ボリュームクレーム

実際のエコシステムでは、システム管理者がPersistentVolumeを作成し、開発者がポッド内で参照されるPersistentVolumeClaimを作成します。PersistentVolumeClaimは、persistentVolumeに必要な最小サイズとアクセスモードを指定することで作成されます。
チャレンジ
上記で作成した永続ボリュームを要求する永続ボリュームクレームを作成します。このクレームは2Giを要求します。永続ボリュームクレームのstorageClassNameが、先ほど作成したpersistentVolumeと同じであることを確認してください。
種類: PersistentVolume apiバージョン: v1 メタデータ: name:app-data
仕様:
accessModes: - ReadWriteManyリソース:
リクエスト :ストレージ :2Gi
ストレージクラス名: 共有
2. 保存してPVCを作成する
njerry191@cloudshell:~ (extreme-clone-2654111)$ kubect1 create -f app-data.yaml
persistentvolumeclaim/app-data が作成されました
3. PVCを見る

```
njerry191@cloudshell:~ (extreme-clone-265411)$ kubectl get pvc
NAME    STATUS    VOLUME    CAPACITY   ACCESS MODES    STORAGECLASS
pv      Bound     pv        512m       RWX             shared
```

4. 最初に作成した PV で何が変わったか確認してみましょう。

```
njerry191@cloudshell:~ (extreme-clone-265411)$ kubectl get pv
NAME    CAPACITY   ACCESS MODES   RECLAIM POLICY   STATUS   CLAIM        STORAGECLASS   REASON   AGE
pv      512m       RWX            Retain           Bound    default/pv   shared         16m
```

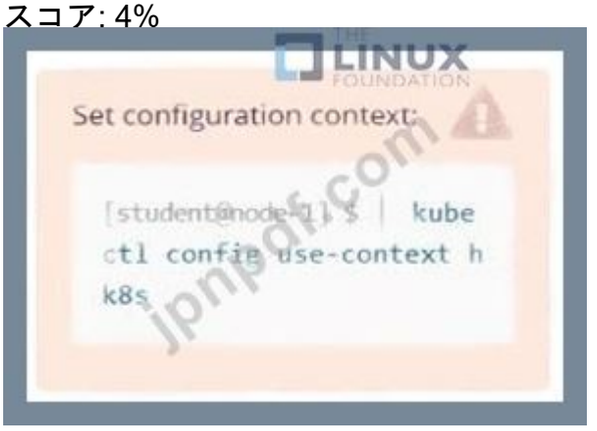
ステータスが 「利用可能」から 「バインド済み」に変更されました。
5. パス /var/app/config を使用して永続ボリューム要求をマウントするために使用される、イメージ nginx を使用して、myapp という名前の新しいポッドを作成します。
主張を展開する
apiVersion: v1 kind: Pod metadata: creationTimestamp: null name: app-data spec: volumes: - name:congigpvc persistentVolumeClaim: claimsName: app-data containers: - image: nginx name: app volumeMounts: - mountPath: "/srv/app-data " name: configpvc

最新問題**: 67**
スコア:7%

タスク
新しいPersistentVolumeClaimを作成する
* 名前: pv-volume
* クラス: csi-hostpath-sc
* 容量: 10Mi
PersistentVolumeClaim をボリュームとしてマウントする新しい Pod を作成します。
* 名前: ウェブサーバー
* 画像: nginx
* マウントパス: /usr/share/nginx/html
新しい Pod がボリュームに対して ReadWriteOnce アクセスを持つように設定します。
最後に、kubectl edit または kubectl patch を使用して、PersistentVolumeClaim を 70Mi の容量まで拡張し、その変更を記録します。
Answer:
解決：
pvc.yaml
ストレージクラス PVC
APIバージョン: v1
種類: PersistentVolumeClaim
メタデータ:
名前: pv-volume
仕様:
アクセスモード:
- 一度だけ読み書き可能
ボリュームモード: ファイルシステム
リソース：
リクエスト:
ストレージ: 10Mi
ストレージクラス名: csi-hostpath-sc
pod-pvc.yaml
APIバージョン: v1
種類: ポッド

メタデータ:
名前: ウェブサーバー
仕様:
コンテナ:
- 名前: ウェブサーバー
画像: nginx
ボリュームマウント:
- マウントパス: "/usr/share/nginx/html"
名前: my-volume
ボリューム:
- 名前: my-volume
永続ボリュームクレーム:
クレーム名: pv-volume
作成
kubectl create -f pod-pvc.yaml
＃編集
kubectl edit pvc pv-volume --record

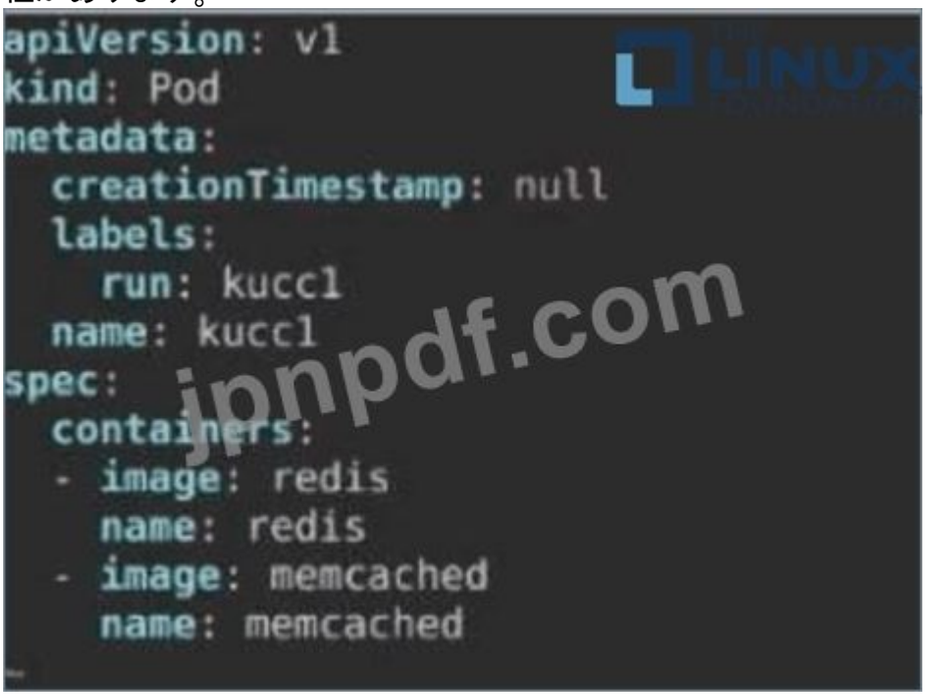
最新問題**: 68**
スコア: 4%



タスク
app-data という名前、容量1Gi、アクセスモードReadOnlyManyの永続ボリュームを作成します。
ボリュームタイプはhostPath、場所は/srv/app-dataです。
Answer:
以下の解決策を参照してください。
説明
解決：
#vi pv.yaml
APIバージョン: v1
種類: 永続ボリューム
メタデータ:
名前: app-config

仕様:
容量：
ストレージ: 1Gi
アクセスモード:
- 読み取り専用
ホストパス:
パス: /srv/app-config
#
kubectl create -f pv.yaml

最新問題**: 69**
次のようにポッドをスケジュールします。
名前: kucc1
アプリコンテナ数: 2
. コンテナ名/イメージ:
* レディス
* メムキャッシュ
Answer:
コンピューターの AI によって生成されたコンテンツのスクリーンショットは不正確である可能性があります。



最新問題**: 70**
デプロイメント redis のサービスとポッドの DNS レコードを取得し、値を /tmp/dnsrecordpod と /tmp/dnsrecord-service に格納します。
A. // ポッドIPを取得する
kubectl get po -o ワイド

// サービス名を取得する
kubectl get svc
// 一時的なポッドを作成し、nslookup コマンドを実行する
注意: POD IP 形式は abcd ではなく abcd である必要があります。
kubectl 実行 busybox --image=busybox:1.28 --restart=Never -
-rm -it --nslookup 192-168-0-69.default.pod >
/tmp/dnsrecord-pod
kubectl run busybox1 --image=busybox:1.26 --restart=Never
--rm -it -- nslookup redis-service > /tmp/dnsrecordservice
//確認する
cat /tmp/dnsrecord-pod
サーバー: 10.2.8.10
アドレス 1: 10.2.0.10 kube-dns.kube system.svc.cluster.local 名前: 192-168-0-69.default.pod アドレス 1: 192.168.0.69 192-166-0-69.redis service.default.svc.cluster.local cat /tmp/dnsrecord-pod サーバー: 10.2.0.10 アドレス 1: 10.2.0.10 kube-dns.kube system.svc.cluster.local 名前: 192-168-0-69.default.pod アドレス 1: 192.168.0.69 192-168-0-69.redis service.default.svc.cluster.local
B. // ポッドIPを取得する
kubectl get po -o ワイド
// サービス名を取得する
kubectl get svc
// 一時的なポッドを作成し、nslookup コマンドを実行する
注意: POD IP 形式は abcd ではなく abcd である必要があります。
kubectl 実行 busybox --image=busybox:1.28 --restart=Never -
-rm -it --nslookup 192-168-0-69.default.pod >
/tmp/dnsrecord-pod
kubectl run busybox1 --image=busybox:1.28 --restart=Never
--rm -it -- nslookup redis-service > /tmp/dnsrecordservice
//確認する
cat /tmp/dnsrecord-pod
サーバー: 10.2.0.10
アドレス 1: 10.2.0.10 kube-dns.kube system.svc.cluster.local 名前: 192-168-0-69.default.pod アドレス 1: 192.168.0.69 192-168-0-69.redis service.default.svc.cluster.local cat /tmp/dnsrecord-pod サーバー: 10.2.0.10 アドレス 1: 10.2.0.10 kube-dns.kube system.svc.cluster.local 名前: 192-168-0-69.default.pod アドレス 1: 192.168.0.69 192-168-0-69.redis service.default.svc.cluster.local
Answer: B (メーッセージを残す**)**

最新問題**: 71**
「my-app」という名のデプロイメントがあり、5つのレプリカでアプリケーションコンテナが実行されています。更新プロセス中は、最大2つのポッドが利用不可になっても問題ないローリング

アップデート戦略を実装する必要があります。また、新しいイメージがDocker Hubリポジトリ「my.org/my-app:latest」にプッシュされるたびに、更新プロセスが自動的にトリガーされるようにする必要があります。

Answer:

以下のステップバイステップの説明付きの解決策を参照してください。

Explanation:

解決策（ステップバイステップ）：

1. デプロイメント YAML を更新します。

- 「レプリカ」を 5 に更新します。
- ローリング更新プロセスを制御できるように、「strategy.rollingUpdate」セクションで「maxUnavailable: 2」と「maxSurge: 0」を定義します。
- デプロイメントが更新されたときにローリング アップデートをトリガーするには、「strategy.type」を「RollingUpdate」に設定します。
- 'spec.template.spec.imagePullPolicy: Always' を追加して、新しいイメージがポッドのローカル キャッシュに存在する場合でも確実にプルされるようにします。

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: my-app
spec:
  replicas: 5
  selector:
    matchLabels:
      app: my-app
  template:
    metadata:
      labels:
        app: my-app
    spec:
      containers:
      - name: my-app
        image: my-org/my-app:latest
        imagePullPolicy: Always
  strategy:
    type: RollingUpdate
    rollingUpdate:
      maxUnavailable: 2
      maxSurge: 0
```


2. デプロイメントの作成: - 'kubectl apply -f my-app.yamP' を使用して、更新された YAML ファイルを適用します。3. デプロイメントの検証: - 'kubectl get deployments my-app' を使用してデプロイメントのステータスを確認し、ロールアウトと更新されたレプリカ数を確認します。4. 自動更新のトリガー: - 新しいイメージを 'my.org/my-app:latest' Docker Hub リポジトリにプッシュします。5. デプロイメントの監視: - 'kubectl get pods -l app=my-app' を使用して、ローリング アップデート プロセス中のポッドの更新を監視します。更新されたイメージを含む新しいポッドが作成されている間に、最大 2 つのポッドが一度に終了されることがわかります。6. 更新の成功の確認: - デプロイメントが完了したら、'kubectl describe deployment my-app' を使用して、'updatedReplicas' フィールドが 'replicas' フィールドと一致していることを確認します。これ

は、更新が成功したことを示します。

最新問題: **72**
「database-deployment」というデプロイメントがあり、データベースコンテナのレプリカが2つあります。ローリングアップデート戦略を実装し、一度に利用できないポッドを最大1つに制限したいと考えています。ただし、データベースコンテナに障害やクラッシュが発生した場合は、自動的に再起動されるようにする必要があります。この再起動ポリシーは、デプロイメント内のデータベースコンテナにのみ適用し、同じポッド内の他のコンテナには影響を与えないようにする必要があります。
Answer:
以下のステップバイステップの説明付きの解決策を参照してください。
Explanation:
解決策（ステップバイステップ）：
1. デプロイメント YAML を更新します。
- 「レプリカ」を 2 に更新します。
- ローリング更新プロセスを制御できるように、strategy.rollingUpdate セクションで 'maxUnavailable: 1' と 'maxSurge: 0' を定義します。
- デプロイメントが更新されたときにローリング アップデートをトリガーするには、「strategy.type」を「RollingUpdate」に設定します。
- 障害が発生したときにデータベースコンテナが自動的に再起動するように、「spec.template.spec.containers[0].restartPolicy: Always」を設定します。

2. デプロイメントの作成: - 'kubectl apply -f database-deployment.yaml' を使用して、更新された YAML ファイルを適用します。3. デプロイメントの検証: - 'kubectl get deployments database-deployment' を使用してデプロイメントのステータスを確認し、ロールアウトと更新されたレプリカ数を確認します。4. 再起動ポリシーのテスト: - ポッドの 1 つ内の 'database' コンテナで障害をトリガーします。これは、コンテナに SIGKILL シグナルを送信するか、コンテナ自体の内部でクラッシュをシミュレートすることで実行できます。- 'database' コンテナが自動的に再起動しますが、同じポッド内の他のコンテナは影響を受けないことを確認します。5. 自動更新のトリガー: - 新しいイメージを 'my-database-image:latest レジストリにプッシュします。6. デプロイメントの監視: - 「kubectl get pods -l app=database」を使用して、ローリング アップデート プロセス中のポッドの更新を監視します。7. 更新の成功の確認: - デプロイメントが完了したら、「kubectl describe deployment database-deployment」を使用して、「updatedReplicas」フィールドが「replicas」フィールドと一致していることを確認します。これは、更新が成功したことを示します。

最新問題: **73**

フロントエンドコンテナのレプリカを 5 つ持つ「frontend-deployment」というデプロイメントが

あります。最大 2 つのポッドが同時に利用不可になることを許容するローリングアップデート戦略を実装する必要があります。また、更新プロセスが指定された 8 分のタイムアウト内に完了することを保証する必要があります。タイムアウト内に更新が完了しなかった場合、デプロイメントは以前のバージョンに戻る必要があります。さらに、フロントエンドコンテナに「post-start」フックを設定し、トラフィックの受け入れを開始する前にアプリケーションの準備状況を確認するヘルスチェックスクリプトを実行します。

Answer:

以下のステップバイステップの説明付きの解決策を参照してください。

Explanation:

解決策（ステップバイステップ）：

1. デプロイメント YAML を更新します。
- 「レプリカ」を 5 に更新します。
- ローリング更新プロセスを制御できるように、「strategy.rollingUpdate」セクションで「maxUnavailable: 2」と「maxSurge: 0」を定義します。
- デプロイメントが更新されたときにローリング アップデートをトリガーするには、「strategy.type」を「RollingUpdate」に設定します。
- 「常に」を設定すると、新しいイメージがプルされるようになります。

ポッドのローカル キャッシュに存在します。

- 更新プロセスのタイムアウトを 8 分に設定するには、「spec.progressDeadlineSeconds: 480」を追加します。
- コンテナがトラフィックの受け入れを開始する前にヘルスチェックスクリプトを実行するスクリプトを定義するために、「spec.template.spec.containers[0].lifecycle.postStart」フックを追加します。

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: frontend-deployment
spec:
  replicas: 5
  selector:
    matchLabels:
      app: frontend
  template:
    metadata:
      labels:
        app: frontend
    spec:
      containers:
      - name: frontend
        image: my-org/frontend:latest
        imagePullPolicy: Always
        lifecycle:
          postStart:
            exec:
              command: ["sh", "-c", "./health-check.sh"]
  strategy:
    type: RollingUpdate
    rollingUpdate:
      maxUnavailable: 2
      maxSurge: 0
  progressDeadlineSeconds: 480
```


2. デプロイメントの作成: - 'kubectl apply -f frontend-deployment.yaml' を使用して、更新された YAML ファイルを適用します。3. デプロイメントの検証: - 'kubectl get deployments frontend-deployment' を使用してデプロイメントのステータスを確認し、ロールアウトと更新されたレプリカ数を確認します。4. 自動更新のトリガー: - 新しいイメージを 'my.org/frontend:latest' Docker Hub リポジトリにプッシュします。5. デプロイメントの監視: - 'kubectl get pods -l app=frontend' を使用して、ローリング更新プロセス中のポッドの更新を監視します。6. タイムアウト超過時のロールバックの確認: - 更新プロセスの完了に 8 分以上かかる場合、デプロイメントは以前のバージョンにロールバックされます。これは、'kubectl describe deployment frontend-deployment' を使用して 'updatedReplicas' フィールドと 'availableReplicas' フィールドを確認することで確認できます。

最新問題: **74**
容量順に並べられたすべての永続ボリュームを一覧表示し、kubectlの出力全体を保存/opt/KUCC00102/volume_list。出力のソートにはkubectl独自の機能を使用し、それ以上の操作は行わないでください。
Answer:
以下の解決策を参照してください。
説明
解決

最新問題: **75**
リテラル値を持つmyconfigmapというconfigmapを作成します
アプリ名=myapp
A. kubectl create cm myconfigmap --from-literal=appname=myapp
// 確認する
または）
kubectl で cm を記述する
B. kubectl create cm myconfigmap --from-literal=appname=myapp
// 確認する
kubectl get cm -o yaml
または）
kubectl で cm を記述する
Answer: (解答を表示する)

最新問題: **76**

pod foo のログを監視し、次の操作を実行します。

* エラーに対応するログ行を抽出

ウェブサイトにアクセスできない

* /opt/KULM00201/foo に書き込む

Answer:

以下の解決策を参照してください。

説明

解決

最新問題: **77**
タスクの重み: 4%

タスク
デプロイメント ウェブサーバーを 3 つのポッドにスケーリングします。
Answer:
解決：

```
student@node-1:~$ kubectl  scale  deploy webserver  --replicas=3
deployment.apps/webserver scaled
student@node-1:~$ kubectl  scale  deploy webserver  --replicas=3
```

最新問題: **78**
次のようにポッドを作成します。
* 名前: mongo
* 使用画像: mongo
* 新しいKubernetes名前空間で
Answer:
以下の解決策を参照してください。
説明
解決

最新問題: **79**
ポッドのIPアドレスを取得する - "nginx-dev"
Answer:
以下の解決策を参照してください。
説明
Kubect1 は po -o ワイドを取得します
JsonPathの使用
kubect1 get pods -o=jsonpath='{range
アイテム[*]}{.metadata.name}{"\t"}{.status.podIP}{"\n"}{end}'

最新問題: **80**

「hello world」をエコーして終了するポッドを作成します。完了したらポッドを自動的に削除します。

Answer:

kubectl run busybox --image=busybox -it --rm --restart=Never -- /bin/sh -c 'echo hello world'
kubectl get po # 「busybox」という名前のポッドは表示されないはずです

最新問題: **81**

pod foo のログを監視し、次の操作を実行します。
エラーに対応するログ行を抽出する
ウェブサイトにアクセスできない
/opt/KULM00201/fooに書き込む

Answer:

解決

最新問題: **82**
ポッド名と開始時刻を「/opt/pod-status」ファイルに出力します。
Answer:
kubect1 でポッドを取得します -o=jsonpath='{range .items[*]}{.metadata.name}{"\t"}
{.status.podIP}{"\n"}{end}'

最新問題: **83**
「hello world」をエコーして終了するポッドを作成します。完了したらポッドを自動的に削除します。以下の解決策をご覧ください。
Answer:
kubectl 実行 busybox --image=busybox -it --rm --restart=Never --
/bin/sh -c 'エコー hello world'
kubectl get po # 「busybox」という名前のポッドは表示されないはずです

最新問題: **84**
次のようにポッドを作成します。
名前: mongo
使用画像: mongo
新しいKubernetes名前空間に「my-website」という名前を付けます
Answer:
以下の解決策を参照してください。

説明
解決
F:\Work\Data Entry Work\Data Entry\20200827\CKA\9 B.JPG

最新問題**: 85**
スコア: 7%
タスク
既存のデプロイメント フロントエンドを再構成し、既存のコンテナ nginx のポート 80/tcp を公開する http という名前のポート仕様を追加します。
コンテナ ポート http を公開する front-end-svc という名前の新しいサービスを作成します。
新しいサービスを設定して、個々のポッドがスケジュールされているノード上の NodePort 経由でそれらのポッドも公開するようにします。
Answer:
以下の解決策を参照してください。
説明
解決：
kubectl get deploy フロントエンド
kubectl edit deploy フロントエンド -o yaml
#http という名前のポート指定
#サービス.yaml
APIバージョン: v1
種類: サービス

メタデータ:
名前: フロントエンドサービス
ラベル:
アプリ: nginx
仕様:
ポート:
- ポート: 80
プロトコル: TCP
名前: http
セレクタ :
アプリ: nginx
タイプ: NodePort
kubectl create -f service.yaml
kubectl get svc
http という名前のポート指定
kubectl expose デプロイメント フロントエンド --name=front-end-svc --port=80 --tarport=80 --type=NodePort

最新問題**: 86**
MySQLデータベースをホストするための永続ボリュームを設定していますが、データの一貫性と可用性を確保したいと考えています。Kubernetesクラスターには複数のノードがあります。このボリュームに使用するボリュームモード、アクセスモード、および再利用ポリシーについて説明してください。
Answer:
以下のステップバイステップの説明付きの解決策を参照してください。
Explanation:
解決策（ステップバイステップ）：
1. ボリュームモード :MySQLデータベースの場合は、「ブロック」ボリュームモードを使用します。ブロックモードでは、ブロックデバイスとしてストレージデバイスに直接アクセスできるため、ストレージに対する低レベルのアクセスと制御が必要なデータベースに最適です。
2. アクセスモード :データの一貫性と可用性を確保するには、「ReadWriteOnce」アクセスモードを使用します。このモードでは、一度に1つのポッドのみがボリュームをマウントできるため、データの破損を防ぎます。
3. 再請求ポリシー：誤ってデータを削除してしまうのを防ぐには、「Retain」再請求ポリシーを使用します。このポリシーは、ポッドが削除されてもボリュームが削除されず、データベースデータが保持されることを保証します。指定された設定を使用したPersistentVolume定義の例を以下に示します。