i created one master instance and 2 worker nodes in our cluster and nothing else.

Now we wanna create pods in replicaset kubernets.
Try creating, and deleting nodes from replicaset only to realise they are redundant and created again, lucidly with commands.

And then try isolating pods from replica set by changing labels.... creating more labels to see nodes.

Then show replicaset scaling - both vertically and horizontally

autoscaling when loads increase and decrease i.e pods created and deleted based on min, max limits.

I'll guide you through creating a ReplicaSet in Kubernetes, managing pods, manipulating labels, scaling, and setting up autoscaling. I'll provide clear commands and explanations for each step, assuming you're working with a cluster that has one master and two worker nodes.

Step 1: Create a ReplicaSet

A ReplicaSet ensures a specified number of pod replicas are running at any given time. Let's create a ReplicaSet with 3 replicas of an Nginx pod.

Create a ReplicaSet YAML file (`nginx-replicaset.yaml`):

```yaml
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: nginx-replicaset
  labels:
    app: nginx
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:latest
        ports:
        - containerPort: 80
```

Apply the ReplicaSet:

```bash
kubectl apply -f nginx-replicaset.yaml
```

Verify the ReplicaSet:

```bash
kubectl get replicaset
```

Output:

```text
NAME              DESIRED   CURRENT   READY   AGE
nginx-replicaset  3         3         3       10s
```

Check the pods:

```bash
kubectl get pods
```

```
temp.sh: line 1: kubectl: command not found
```

Output (example):

```text
NAME                     READY   STATUS    RESTARTS   AGE
nginx-replicaset-abc12   1/1     Running   0          15s
nginx-replicaset-def34   1/1     Running   0          15s
nginx-replicaset-ghi56   1/1     Running   0          15s
```

Step 2: Delete Pods to Observe Redundancy

The ReplicaSet maintains the desired number of pods. If you delete a pod, the ReplicaSet will create a new one to meet the `replicas: 3` specification.

Delete a pod:

```bash
kubectl delete pod nginx-replicaset-abc12
```

Check pods again:

```bash
kubectl get pods
```

```
temp.sh: line 1: kubectl: command not found
```

Output (example):

```text
NAME                     READY   STATUS    RESTARTS   AGE
nginx-replicaset-def34   1/1     Running   0          30s
nginx-replicaset-ghi56   1/1     Running   0          30s
nginx-replicaset-jkl78   1/1     Running   0          5s
```

The ReplicaSet automatically creates a new pod (`nginx-replicaset-jkl78`) to replace the deleted one.

Step 3: Isolate Pods by Changing Labels

Pods are managed by a ReplicaSet based on their labels matching the `selector.matchLabels` . If you change a pod's label so it no longer matches, the ReplicaSet will treat it as outside its control and create a new pod to maintain the desired replica count.

View current pod labels:

```bash
kubectl get pods --show-labels
```

Output (example):

```text
NAME                    READY   STATUS    RESTARTS   AGE   LABELS
nginx-replicaset-def34  1/1     Running   0          1m    app=nginx
nginx-replicaset-ghi56  1/1     Running   0          1m    app=nginx
nginx-replicaset-jkl78  1/1     Running   0          30s   app=nginx
```

Change the label of one pod:

```bash
kubectl label pod nginx-replicaset-def34 app=nginx-modified --overwrite
```

Verify the label change:

```bash
kubectl get pods --show-labels
```

Output (example):

```text
NAME                    READY   STATUS    RESTARTS   AGE   LABELS
nginx-replicaset-def34  1/1     Running   0          2m    app=nginx-
modified
```

```
nginx-replicaset-ghi56   1/1     Running    0          2m     app=nginx
nginx-replicaset-jkl78   1/1     Running    0          1m     app=nginx
nginx-replicaset-mno90   1/1     Running    0          10s    app=nginx
```

The ReplicaSet no longer recognizes the pod with `app=nginx-modified` and creates a new pod (`nginx-replicaset-mno90`) to maintain 3 replicas with `app=nginx` .

Add more labels to a pod:

```bash
kubectl label pod nginx-replicaset-ghi56 env=prod tier=frontend
```

Verify new labels:

```bash
kubectl get pods --show-labels
```

Output (example):

```text
NAME                     READY   STATUS     RESTARTS   AGE    LABELS
nginx-replicaset-def34   1/1     Running    0          3m     app=nginx-
modified
nginx-replicaset-ghi56   1/1     Running    0          3m
app=nginx,env=prod,tier=frontend
nginx-replicaset-jkl78   1/1     Running    0          2m     app=nginx
nginx-replicaset-mno90   1/1     Running    0          1m     app=nginx
```

Adding extra labels (`env=prod` , `tier=frontend`) doesn't affect the ReplicaSet unless the `app=nginx` label is removed.

Step 4: ReplicaSet Scaling

Horizontal Scaling (Adjusting Replicas)

Horizontal scaling changes the number of pod replicas.

Scale up to 5 replicas:

```bash
kubectl scale replicaset nginx-replicaset --replicas=5
```

Verify:

```bash
kubectl get replicaset
```

Output:

```text
NAME               DESIRED   CURRENT   READY   AGE
nginx-replicaset   5         5         5       5m
```

Check pods:

```bash
kubectl get pods
```

```
temp.sh: line 1: kubectl: command not found
```

Output (example):

```text
NAME                     READY   STATUS    RESTARTS   AGE
nginx-replicaset-def34   1/1     Running   0          5m
nginx-replicaset-ghi56   1/1     Running   0          5m
nginx-replicaset-jkl78   1/1     Running   0          4m
nginx-replicaset-mno90   1/1     Running   0          3m
nginx-replicaset-pqr12   1/1     Running   0          10s
```

Scale down to 2 replicas:

```bash
```

```
kubectl scale replicaset nginx-replicaset --replicas=2
```

Verify:

```bash
kubectl get replicaset
```

Output:

```text
NAME               DESIRED   CURRENT   READY   AGE
nginx-replicaset   2         2         2       6m
```

Check pods:

```bash
kubectl get pods
```

```
temp.sh: line 1: kubectl: command not found
```

Output (example):

```text
NAME                     READY   STATUS    RESTARTS   AGE
nginx-replicaset-ghi56   1/1     Running   0          6m
nginx-replicaset-jkl78   1/1     Running   0          5m
```

The pod with `app=nginx-modified` remains because it's no longer managed by the ReplicaSet.

Vertical Scaling (Adjusting Resources)

Vertical scaling involves changing the CPU/memory requests or limits for pods. You need to edit the ReplicaSet's pod template.

Edit the ReplicaSet:

```bash
kubectl edit replicaset nginx-replicaset
```

Add resource requests/limits under `spec.template.spec.containers`:

```yaml
spec:
  containers:
  - name: nginx
    image: nginx:latest
    ports:
    - containerPort: 80
    resources:
      requests:
        cpu: "100m"
        memory: "128Mi"
      limits:
        cpu: "200m"
        memory: "256Mi"
```

Save and exit. Kubernetes will roll out new pods with the updated resources, replacing the old ones.

Verify new pods:

```bash
kubectl get pods
```

```
temp.sh: line 1: kubectl: command not found
```

Output (example):

```text
NAME                    READY   STATUS    RESTARTS   AGE
nginx-replicaset-stu34  1/1     Running   0          10s
nginx-replicaset-vwx56  1/1     Running   0          10s
```

Check resource allocation:

```bash
kubectl describe pod nginx-replicaset-stu34
```

Look for the `Limits` and `Requests` sections in the output.

Step 5: Autoscaling with Horizontal Pod Autoscaler (HPA)

The Horizontal Pod Autoscaler (HPA) automatically scales the number of pods based on metrics like CPU utilization.

Create an HPA:

```bash
kubectl autoscale replicaset nginx-replicaset --min=2 --max=5 --cpu-percent=80
```

Verify the HPA:

```bash
kubectl get hpa
```

Output:

```text
NAME               REFERENCE                          TARGETS   MINPODS   MAXPODS   REPLICAS   AGE
nginx-replicaset   ReplicaSet/nginx-replicaset   0%/80%    2         5         2          10s
```

Simulate load to trigger scaling:
Use a tool like `hey` or `ab` to generate HTTP requests to the Nginx pods. First, expose the pods via a Service:

Create a Service (`nginx-service.yaml`):

```yaml
apiVersion: v1
kind: Service
metadata:
  name: nginx-service
spec:
  selector:
    app: nginx
  ports:
  - protocol: TCP
    port: 80
    targetPort: 80
  type: ClusterIP
```

Apply the Service:

```bash
kubectl apply -f nginx-service.yaml
```

Generate load (example using `hey`):

```bash
hey -n 10000 -c 100 http://<cluster-ip>:80
```

Replace `<cluster-ip>` with the Service's ClusterIP (find it with `kubectl get svc`).

Monitor HPA:

```bash
kubectl get hpa --watch
```

As CPU usage exceeds 80%, the HPA scales up pods (up to 5). When the load decreases, it scales down (to a minimum of 2).

Example output during scaling:

```text
```

```
NAME              REFERENCE                    TARGETS   MINPODS
MAXPODS   REPLICAS   AGE
nginx-replicaset  ReplicaSet/nginx-replicaset  85%/80%   2         5
4         5m
```

Stop the load and observe scale-down:
After stopping the load, the HPA will reduce the number of pods back to the minimum (2) after a cooldown period (default 5 minutes).

Notes

- **Cluster Setup**: Ensure your cluster (1 master, 2 workers) has enough resources to handle the pods and load testing.
- **Metrics Server**: HPA requires the Metrics Server to be installed for CPU-based scaling (`kubectl top` should work).
- **Load Testing**: If `hey` isn't installed, use alternatives like `ab` or a simple `curl` loop, or deploy a load generator pod.
- **Label Isolation**: Pods with modified labels become "orphaned" and persist unless manually deleted.
- **Scaling Limits**: Your two worker nodes may limit the number of pods; ensure sufficient capacity for scaling.

Let me know if you need help with any specific step or additional configurations!