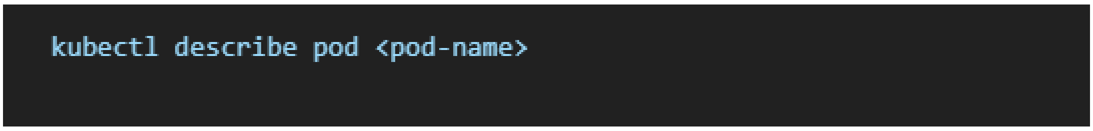
My Kubernetes Troubleshooting Checklist

✓ Describe the Pod in Detail



```
kubectl describe pod <pod-name>
```

Command Break Down

- kubectl → The Kubernetes CLI tool.
- describe → Fetches detailed information about the specified resource.
- pod <pod-name> → Specifies the pod whose details you want to examine.
- When this command is used, detailed information on the pod is produced, including:

  Namespace, Labels, and Pod Name: Namespace, annotations, and labels are examples of basic metadata.
- Pod Status: Indicates if the pod has succeeded, failed, pending, or is running.

  Displays current happenings that have an impact on the pod.
- Node Assignment: Indicates which worker Kubernetes node the pod is operating on.

  Details on the containers in the pod, including picture versions, are listed here.

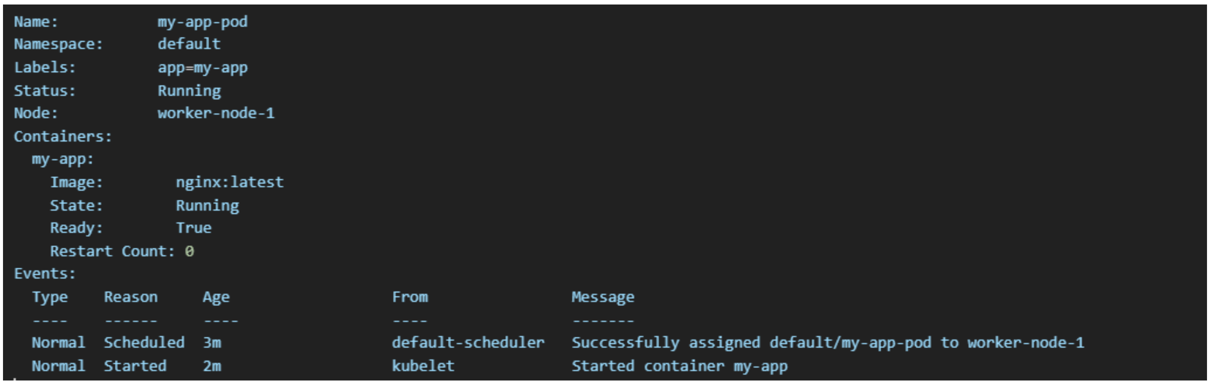
  Variables related to the container environment (env).

  Requests for and limitations on CPU and memory resources.
- Volume Mounts: Shows the pod's internal persistent storage mounts.

  Current Occurrences & Mistakes

  Displays errors, warnings, and other pod-related system events.

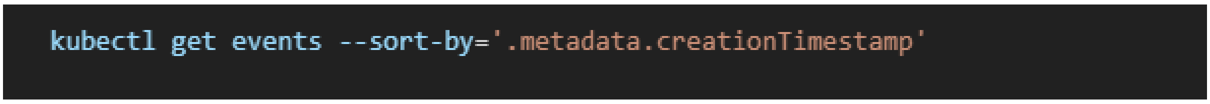
Expected Output :



```
Name:           my-app-pod
Namespace:      default
Labels:         app=my-app
Status:         Running
Node:           worker-node-1
Containers:
  my-app:
    Image:          nginx:latest
    State:          Running
    Ready:          True
    Restart Count: 0
Events:
  Type    Reason     Age                  From               Message
  ----    ------     ----                 ----               -------
  Normal  Scheduled  3m                   default-scheduler  Successfully assigned default/my-app-pod to worker-node-1
  Normal  Started    2m                   kubelet            Started container my-app
```

This often gives more clues than logs: failed mounts, image pull issues, liveness probe failures, or event errors.

✓ Check Events at the Namespace Level



```
kubectl get events --sort-by='.metadata.creationTimestamp'
```

Command Break Down

kubectl get events → Fetches all events occurring within the cluster.

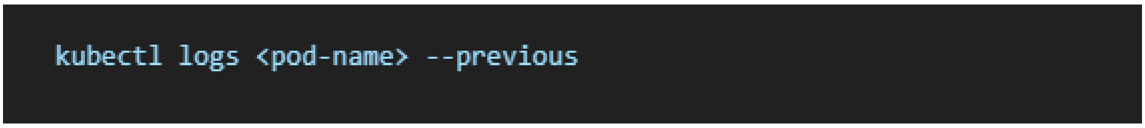
--sort-by='.metadata.creationTimestamp' → Sorts the events from oldest to newest based on their creation time.

✓ Why Use This Command?

- Examine historical occurrences in chronological order.
- Examine what occurred before to a failure in order to troubleshoot cluster concerns.
- Keep an eye on resource fluctuations, node activity, and pod scheduling.

Found a CrashLoopBackOff event caused by a missing secret volume mount. A small misconfiguration, big impact.

✓ Logs From Previous Container State



```
kubectl logs <pod-name> --previous
```

Command Break Down

- kubectl logs → Retrieves log output from a running pod.

- <pod-name> → Specifies the name of the pod whose logs you want to view.
- --previous → Shows logs from the last terminated container rather than the current running instance.
    - ✓ When Should You Use This Command?
- Investigating pod failures → See logs from the previous instance before it crashed or restarted.
- Debugging container restarts → Helps understand why the container terminated unexpectedly.
- Tracking short-lived jobs → Useful for pods running batch jobs that complete quickly.

    - ✓ If a pod has multiple containers, specify the container name:

```
kubectl logs my-app-pod -c my-container --previous
```

    - ✓ Additional Debugging Commands

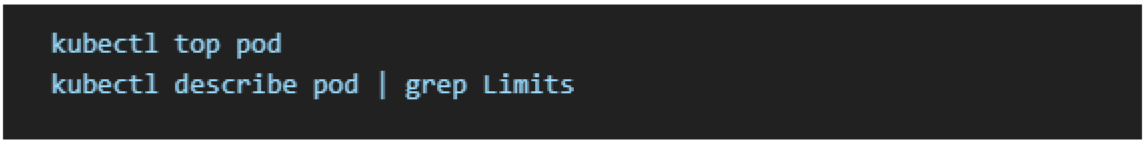
If you suspect a crash loop or failure, combine with:

```
kubectl get pod my-app-pod -o wide
```

Helps identify restart counts, error events, and container lifecycle issues.

Captured a stack trace from the last crash that pointed to a missing environment variable.

- ✓ Check Resource Limits

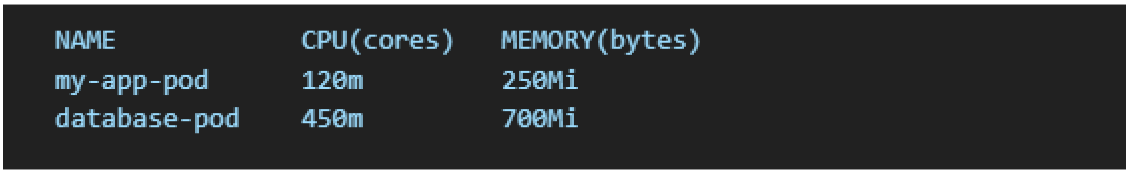


```
kubectl top pod
kubectl describe pod | grep Limits
```

```
kubectl top pod
```

Example Output

- CPU usage → Measured in millicores (m), where 1000m = 1 core.
- Memory usage → Displayed in MB (Mi), GB (Gi).

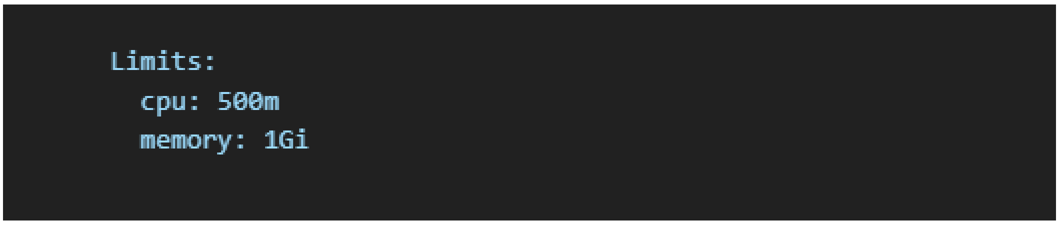


```
NAME            CPU(cores)   MEMORY(bytes)
my-app-pod      120m         250Mi
database-pod    450m         700Mi
```

kubectl describe pod <pod-name> | grep Limits

Example Output

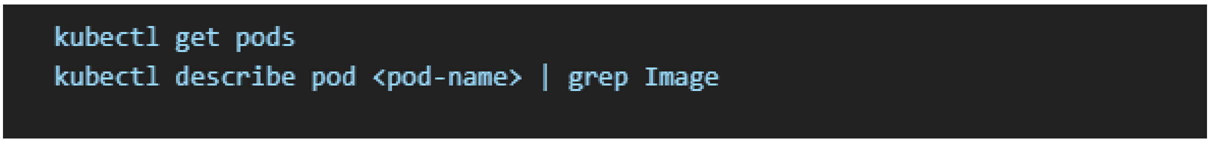
- ➢ CPU Limit → The pod is restricted to 500 millicores (m).
- ➢ Memory Limit → Cannot exceed 1 GiB (Gi).



```
    Limits:
      cpu: 500m
      memory: 1Gi
```

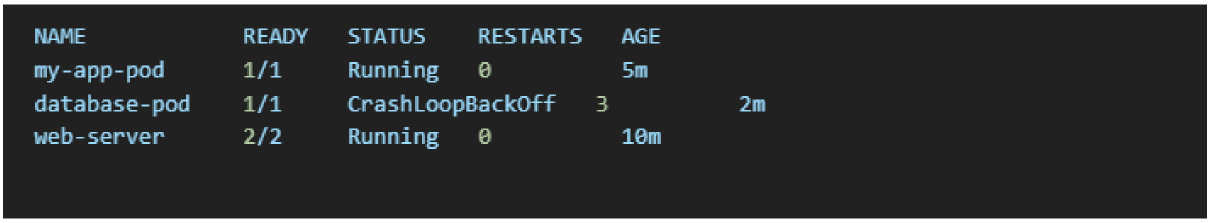
Found a container getting killed by OOM (out of memory) due to strict memory limits.

✓ Check ImagePull Errors



```
kubectl get pods
kubectl describe pod <pod-name> | grep Image
```

Example Output

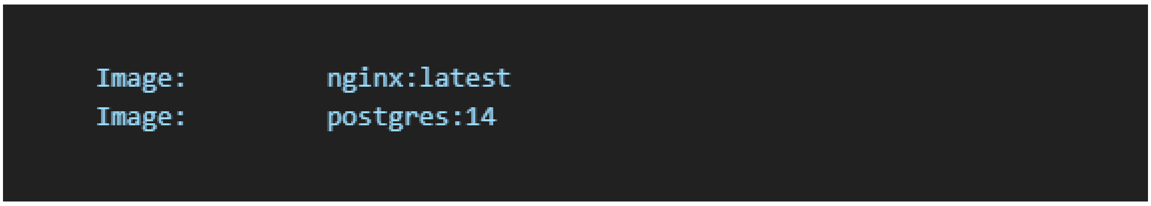


```
NAME            READY   STATUS     RESTARTS    AGE
my-app-pod      1/1     Running    0           5m
database-pod    1/1     CrashLoopBackOff    3           2m
web-server      2/2     Running    0           10m
```

- ➢ NAME → Pod name.
- ➢ READY → Number of containers running vs. expected.
- ➢ STATUS → Running, Pending, Failed, CrashLoopBackOff, etc.
- ➢ RESTARTS → Number of times the pod restarted.
- ➢ AGE → How long the pod has been running.

Useful for monitoring pod health & deployment status.

Example Output

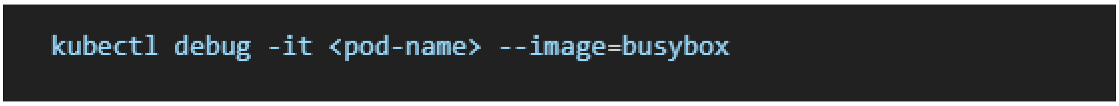


```
Image:         nginx:latest
Image:         postgres:14
```

- Displays container images (nginx:latest, postgres:14) used within the pod. Useful for debugging version mismatches or confirming correct image deployment.

Sometimes it’s as simple as a wrong image tag or private registry permission issue.

✓ Debug Using Ephemeral Containers



```
kubectl debug -it <pod-name> --image=busybox
```

- kubectl debug → Creates a temporary debug container inside a pod.
- -it → Starts an interactive terminal session, allowing direct access to the pod.
- <pod-name> → Specifies the pod you want to debug.
- --image=busybox → Uses the busybox image as the debug container (lightweight shell tools).

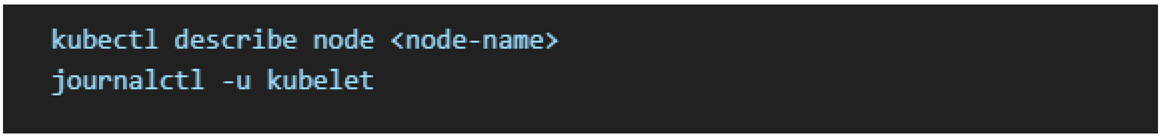
Run tools like curl, nslookup, or cat inside the pod’s namespace to test connectivity and environment variables.

Example Usage

Attaches an interactive BusyBox shell to the my-app-pod pod.

Now you can run troubleshooting commands within the pod.

✓ Dive into Node Issues



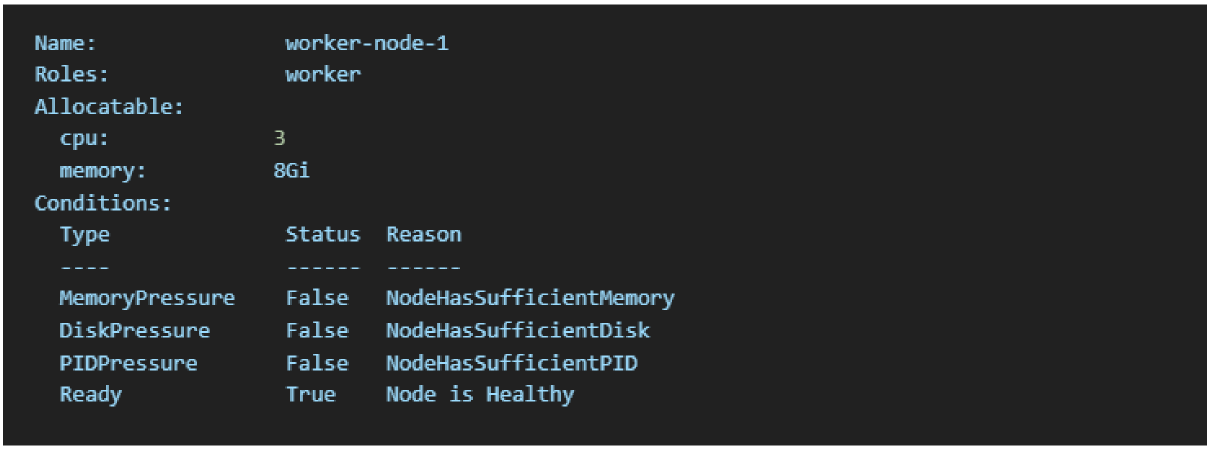
```
kubectl describe node <node-name>
journalctl -u kubelet
```

kubectl describe node <node-name>

Retrieves detailed information about a specific node in the Kubernetes cluster, including:

- Node status (Ready, NotReady, etc.).
- Allocated resources (CPU, Memory).
- Running pods on that node.
- Conditions affecting the node (DiskPressure, MemoryPressure, etc.).

Expected Output



```
Name:                   worker-node-1
Roles:                  worker
Allocatable:
  cpu:                 3
  memory:              8Gi
Conditions:
  Type                  Status  Reason
  ----                  ------  ------
  MemoryPressure        False   NodeHasSufficientMemory
  DiskPressure          False   NodeHasSufficientDisk
  PIDPressure           False   NodeHasSufficientPID
  Ready                 True    Node is Healthy
```

journalctl -u kubelet

✓ Why is it necessary?

retrieves the logs for the Kubelet service, which is in charge of a node's pod and container management.

✓ Easy Say:

Use kubectl describe node to check the node's health.
Examine problems with pod placement (kubectl describe node).
Use journalctl -u kubelet to debug Kubelet errors.
Use journalctl -u kubelet -f to view live Kubelet logs.
Even if your YAML is flawless, pods may behave up due to disk strain or network plugin issues.