How do you debug a failing Pod?

1. Check Pod Status

Run:

```
sh
---
kubectl get pods
```

Look at the `STATUS` column:

- **Pending**: Check if there are insufficient resources.
- **CrashLoopBackOff**: The container is repeatedly failing.
- **Error**: The container encountered a runtime error.
- **Completed**: The job completed successfully.

2. Describe the Pod

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

- Check the **Events** section for scheduling issues, missing images, or resource limits.
- Look at **State and Exit Code** under `Containers:` for clues about why it's failing.

3. Check Logs

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

If the pod has multiple containers:

```
sh
---
kubectl logs <pod-name> -c <container-name>
```

For a terminated container:

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

4. Check for Node Issues

If the pod is not getting scheduled, check the node:

```
sh
---
kubectl get nodes
kubectl describe node <node-name>
```

5. Check Events in the Namespace

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

Look for failures related to scheduling, resource limits, or image pull errors.

6. Exec into the Container

If the pod is running but misbehaving:

```
sh
---
kubectl exec -it <pod-name> -- /bin/sh  # or /bin/bash
```

Then manually inspect logs, processes, and configurations.

7. Check Resource Requests and Limits

If the pod is OOMKilled (Out of Memory):

```
sh
---
kubectl describe pod <pod-name> | grep -i oom
```

Check for `resources.requests` and `resources.limits` in your YAML file.

8. Check if the Image is Available

```
sh
---
kubectl describe pod <pod-name> | grep -i "failed to pull image"
```

Ensure the image exists and is accessible.

9. Check Network Issues

If the pod cannot communicate:

```
sh
---
kubectl get svc
kubectl get endpoints
```

Use `nslookup` or `curl` inside the pod to test connectivity.

10. Delete and Restart the Pod

```
sh
---
kubectl delete pod <pod-name>
```

If it's part of a Deployment, ReplicaSet, or StatefulSet, it should restart automatically.

Step 1: Create a Failing Pod

We'll create a Pod with:

- A non-existent image (`invalid-image:latest`) to simulate an image pull failure.

Save this YAML as `failing-pod.yaml`:

```
yaml
---
apiVersion: v1
kind: Pod
metadata:
  name: failing-pod
spec:
  containers:
  - name: broken-container
    image: invalid-image:latest  # This image does not exist
    command: ["sleep", "1000"]
```

Apply the file:

```
sh
---
kubectl apply -f failing-pod.yaml
```

Step 2: Check Pod Status

Run:

```
sh
---
kubectl get pods
```

You'll see something like:

```
nginx
---
NAME          READY   STATUS             RESTARTS   AGE
failing-pod   0/1     ImagePullBackOff   0          1m
```

The `STATUS` is `ImagePullBackOff`, indicating a problem pulling the container image.

Step 3: Describe the Pod

Run:

```
sh
---
```

```
kubectl describe pod failing-pod
```

You'll see:

```
vbnet
---
Events:
  Type     Reason          Age    From               Message
  ----     ------          ----   ----               -------
  Normal   Scheduled       2m     default-scheduler  Successfully assigned
default/failing-pod to node-1
  Normal   BackOff         1m     kubelet            Back-off pulling image
"invalid-image:latest"
  Warning  Failed          1m     kubelet            Error: ImagePullBackOff
  Warning  Failed          1m     kubelet            Failed to pull image
"invalid-image:latest": rpc error: code = NotFound desc = failed to pull and
unpack image "invalid-image:latest"
```

Key issue:
🚨 **"Failed to pull image 'invalid-image:latest'"** → The image doesn't exist.

If a Pod is stuck in the ImagePullBackOff state, what is the likely issue?

A) Did not login to docker hub so could not pull the image
B) The container is using too much memory
C) The image specified does not exist or is not accessible
D) The Pod does not have enough CPU resources

- **ImagePullBackOff** indicates that Kubernetes is unable to pull the specified container image. This could happen because:
- The image name or tag is incorrect.
- The image does not exist in the specified registry.
- The registry is inaccessible due to network issues or authentication problems.
- The credentials for pulling the image (e.g., from a private registry) are missing or incorrect.

So A and C

What is the first command you should run to check the status of a failing Pod?

A) kubectl logs <pod-name>
B) kubectl get pods

C) kubectl exec -it <pod-name> -- /bin/sh
D) kubectl delete pod <pod-name>

Answer: B) kubectl get pods

How do you get detailed information, including events and errors, about a failing Pod?
A) kubectl logs <pod-name>
B) kubectl describe pod <pod-name>
C) kubectl exec -it <pod-name> -- /bin/sh
D) kubectl get events

B) kubectl describe pod <pod-name>
Explanation:

- **kubectl describe pod <pod-name>** provides detailed information about the Pod, including its configuration, status, and events related to the Pod (e.g., scheduling, pulling images, container failures, etc.). This is the best way to diagnose why a Pod is failing.