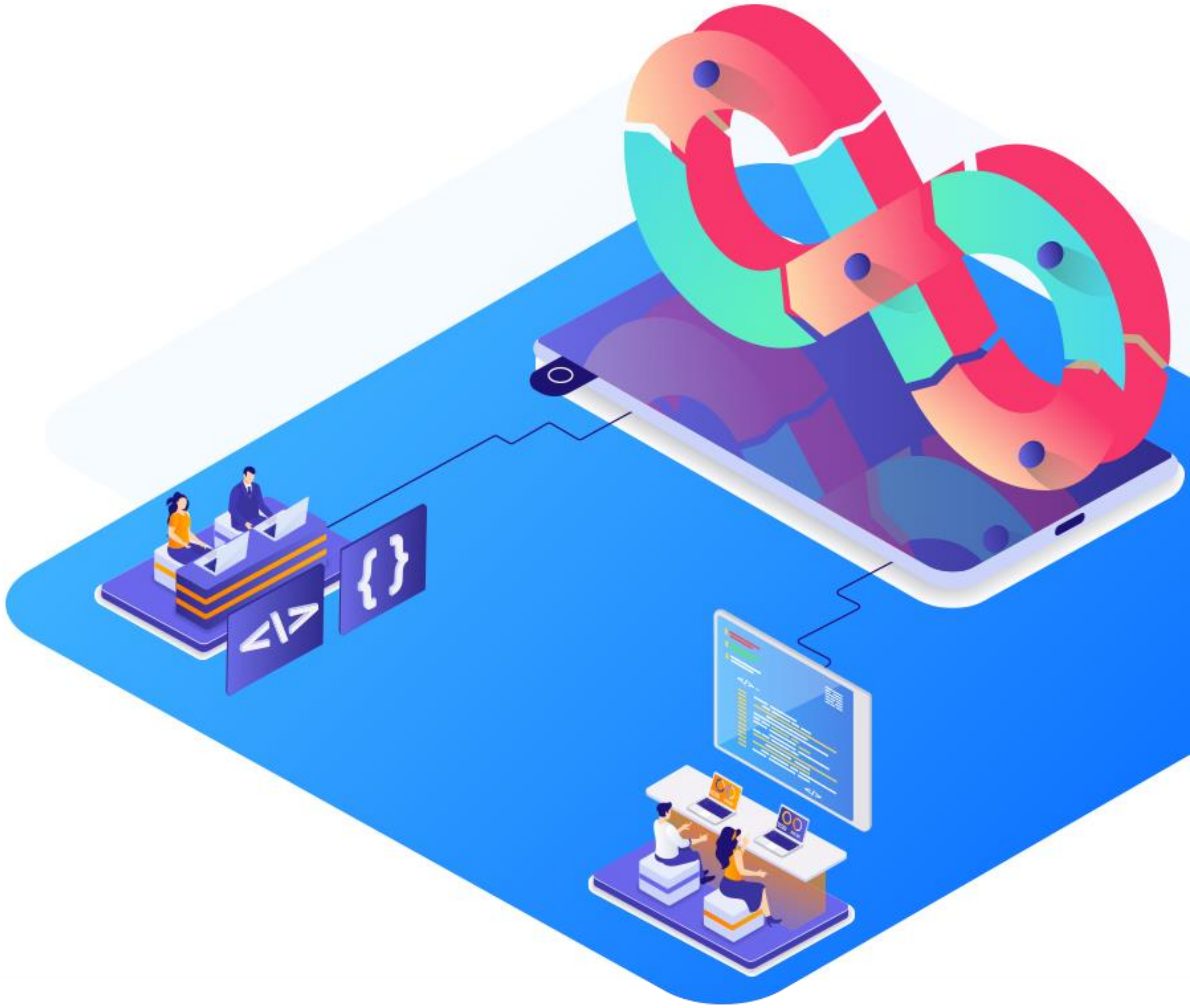
Container Orchestration Using Kubernetes



Troubleshooting and Kubernetes Case Studies

Learning Objectives

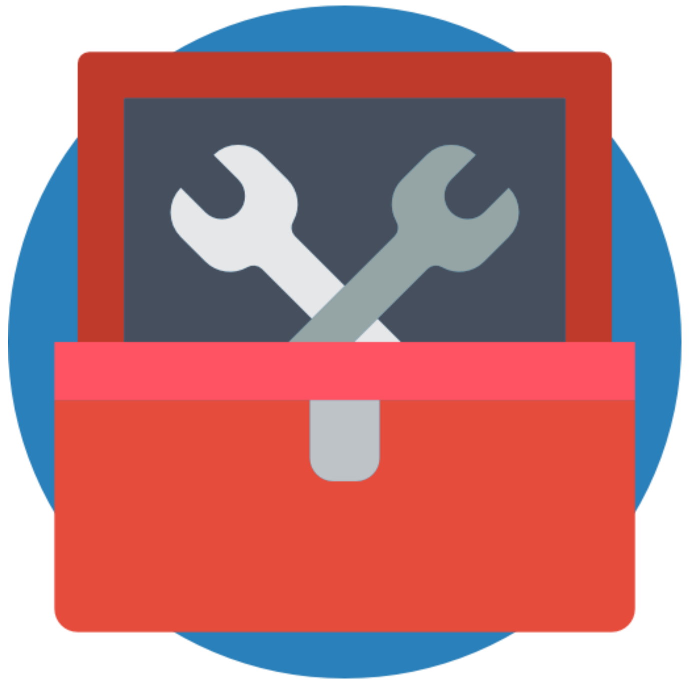
By the end of this lesson, you will be able to:

- Analyze a Kubernetes cluster to ensure optimal performance and reliability
- Configure options in Kubernetes cluster logging architecture for customizing log collection and storage
- Analyze cluster, node-level, and container logs to enhance Kubernetes operations
- Identify and resolve application issues, ensuring functionality and reliability of the software
- Analyze the performance of the application for optimizing resource utilization

Overview of Troubleshooting in Kubernetes

Troubleshooting

It is the process of identifying and resolving issues that may occur at any layer or component within a Kubernetes cluster.



It involves finding the root cause of a failure and taking specific steps to recover from the failure.

Elements for Troubleshooting

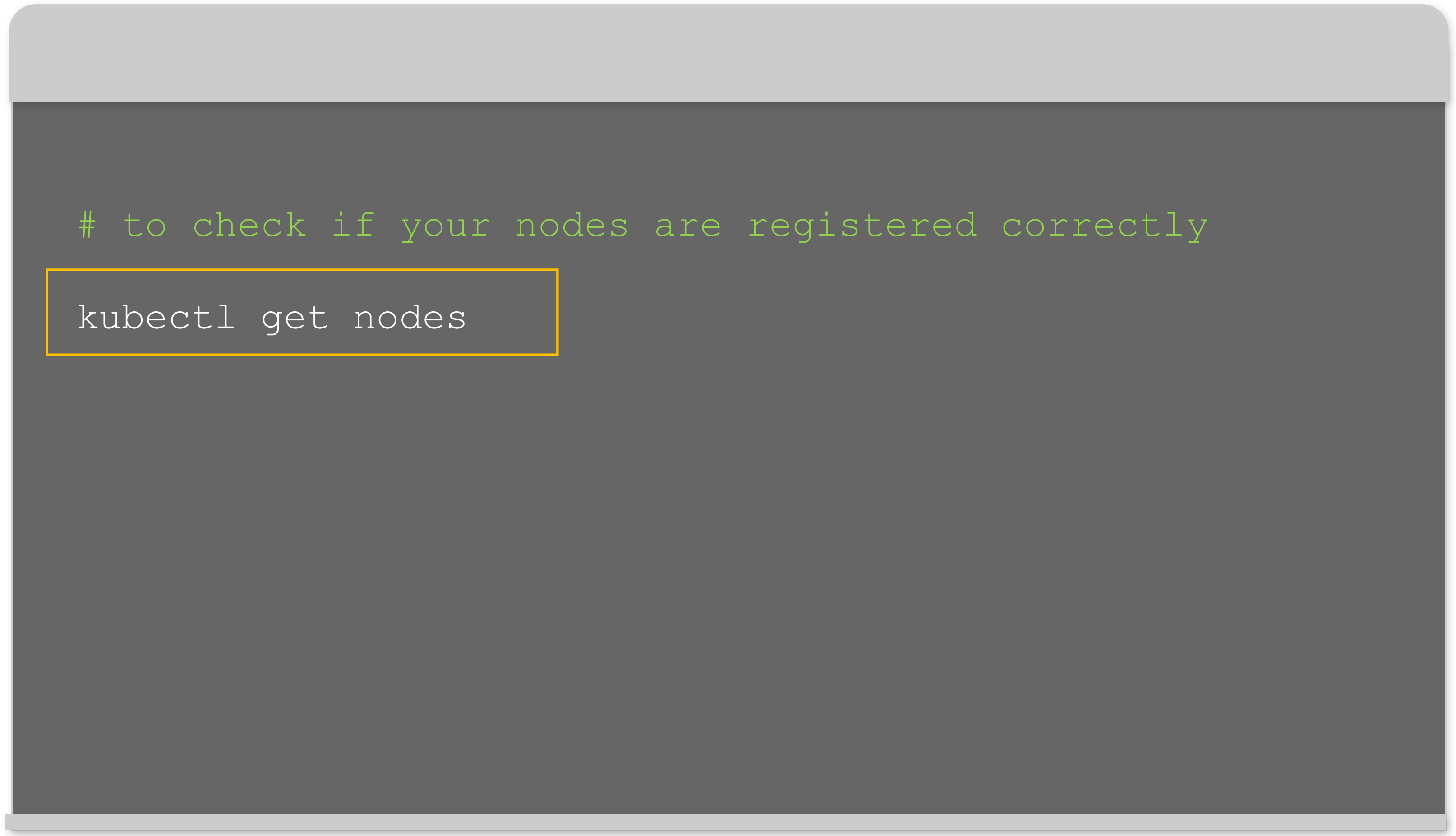
They support debugging and troubleshooting for the administrators to analyze and identify issues.

The elements are:

- Kubernetes cluster
- Containerized applications

Kubernetes Cluster: Debugging a Cluster

A cluster is a set of interconnected nodes, including a master node and one or more worker nodes that work together to manage and orchestrate containerized applications.

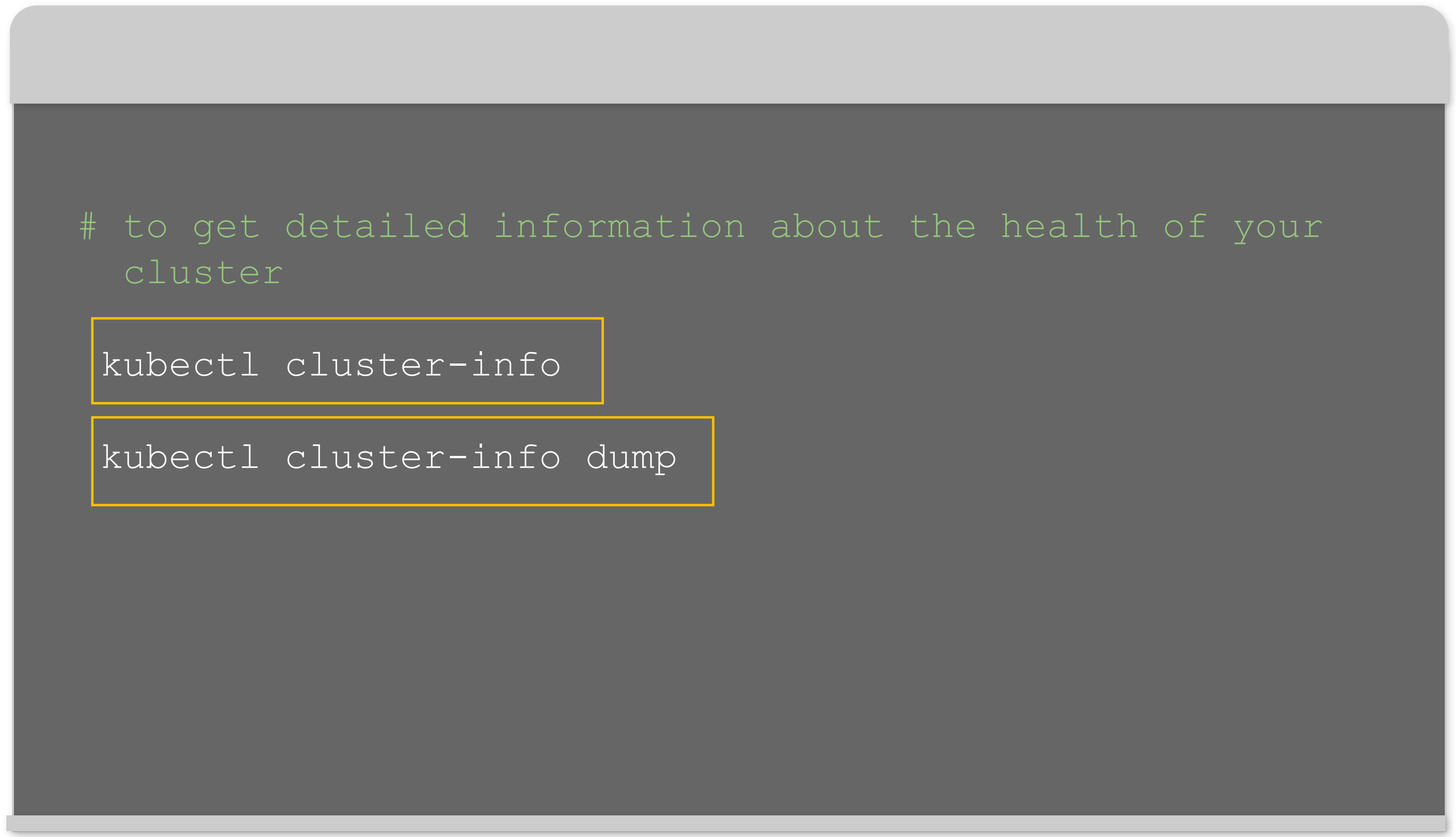


```
# to check if your nodes are registered correctly

kubectl get nodes
```

Ensure all nodes are registered correctly to debug a cluster. It is crucial that each node is in the **Ready** state.

Kubernetes Cluster: Check the Health of the Cluster

The health of the cluster can be checked by executing the following commands:



```
# to get detailed information about the health of your
  cluster

kubectl cluster-info

kubectl cluster-info dump
```

Locations of Log Files on Master Node

A thorough investigation of cluster issues requires analyzing log files located on the master node. These logs provide crucial insights into the performance and behavior of the API server and the scheduler.

/var/log/kube-apiserver.log	Contains logs for the API server, which is responsible for handling requests to the Kubernetes API
/var/log/kube-scheduler.log	Contains logs for the scheduler, which is responsible for making scheduling decisions for the cluster

Locations of Log Files on Master Node

/var/log/kube-controller-manager.log	Contains logs for the controller manager, which is responsible for managing replication controllers
/var/log/kubelet.log	Contains logs for the kubelet, which is responsible for running containers on the node
/var/log/kube-proxy.log	Contains logs for the kube-proxy, which is responsible for service load balancing within the cluster

Root Causes of Cluster Failures

Following are the root causes of cluster failure:

Shutdown of VMs

Network partition within the cluster or between the cluster and users

Crashes in Kubernetes software

Operator error

Data loss or unavailability of persistent storage

Cluster Failure Scenarios

Following are the specific cluster failure scenarios:

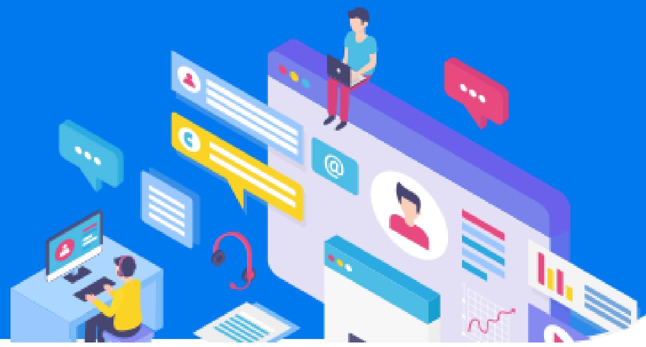
1. API server crash
2. APE backing storage loss
3. Supporting services crash
4. Shutdown of an individual node
5. Network partition failure
6. Kubelet software fault
7. Cluster operator error

Mitigations for Cluster Failures

Following are the mitigations for cluster failures:

- Use IaaS provider's automatic VM restarting feature
- Use IaaS provider's reliable storage
- Use high availability configuration
- Create snapshots of apiserver PDs or EBS volumes periodically
- Use Replication Controller and services
- Design apps to tolerate unexpected restarts

Assisted Practice



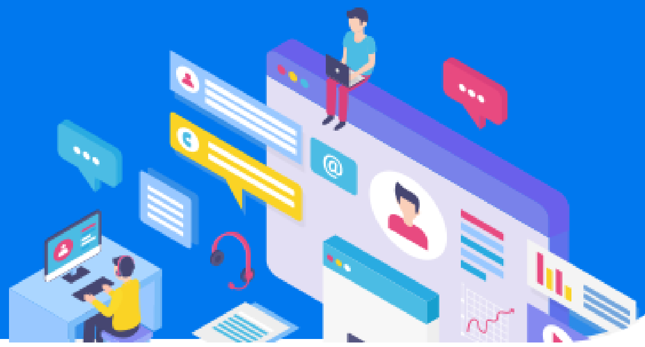
Troubleshooting Kubernetes Cluster **Duration: 10 Min.**

Problem statement:
You have been asked to troubleshoot Kubernetes clusters by utilizing diagnostic commands.

Outcome:
By the end of this demo, you will be able to troubleshoot Kubernetes clusters by utilizing diagnostic commands.

Note: Refer to the demo document for detailed steps

Assisted Practice: Guidelines



Steps to be followed:

1. Troubleshoot using dumps

Quick Check

You are a Kubernetes administrator responsible for your organization's cluster. A developer reported issues with unresponsive services, so you decide to check the cluster's overall health before proceeding.
Which command should you execute to obtain detailed information about the health and status of your Kubernetes cluster?

A. kubectl get nodes

B. kubectl logs

C. kubectl cluster-info

D. kubectl describe pod

Kubernetes Cluster Logging Architecture

Application Logs

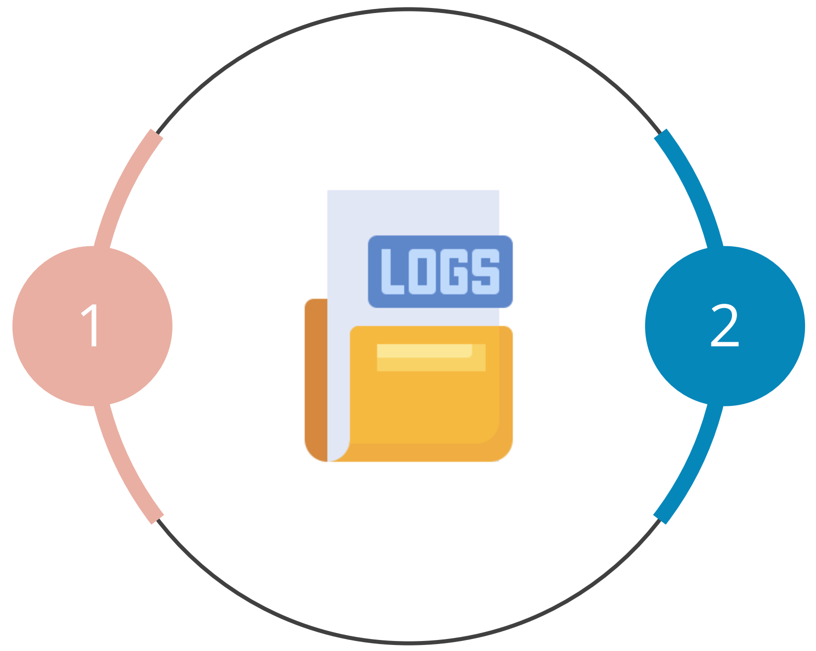
They help to understand the inside of an application.

Most modern applications have a logging mechanism that helps to:

- Debug problems
- Monitor cluster activity

Methods for Logging

The most commonly used logging methods for applications that use containers are:



Write to standard output stream

Write to standard error stream

Cluster-Level Logging Architecture

It ensures that logs are stored independently of the containers, pods, or nodes in the cluster. It enables reliable log management, even when components within the cluster encounter issues.

Cluster-level logging architecture:

Enables access to application logs even if a node dies, a pod gets evicted, or a container crashes, ensuring critical data remains available for troubleshooting

Requires a separate backend to store, analyze, and query logs, as logs are managed outside of the cluster's core resources

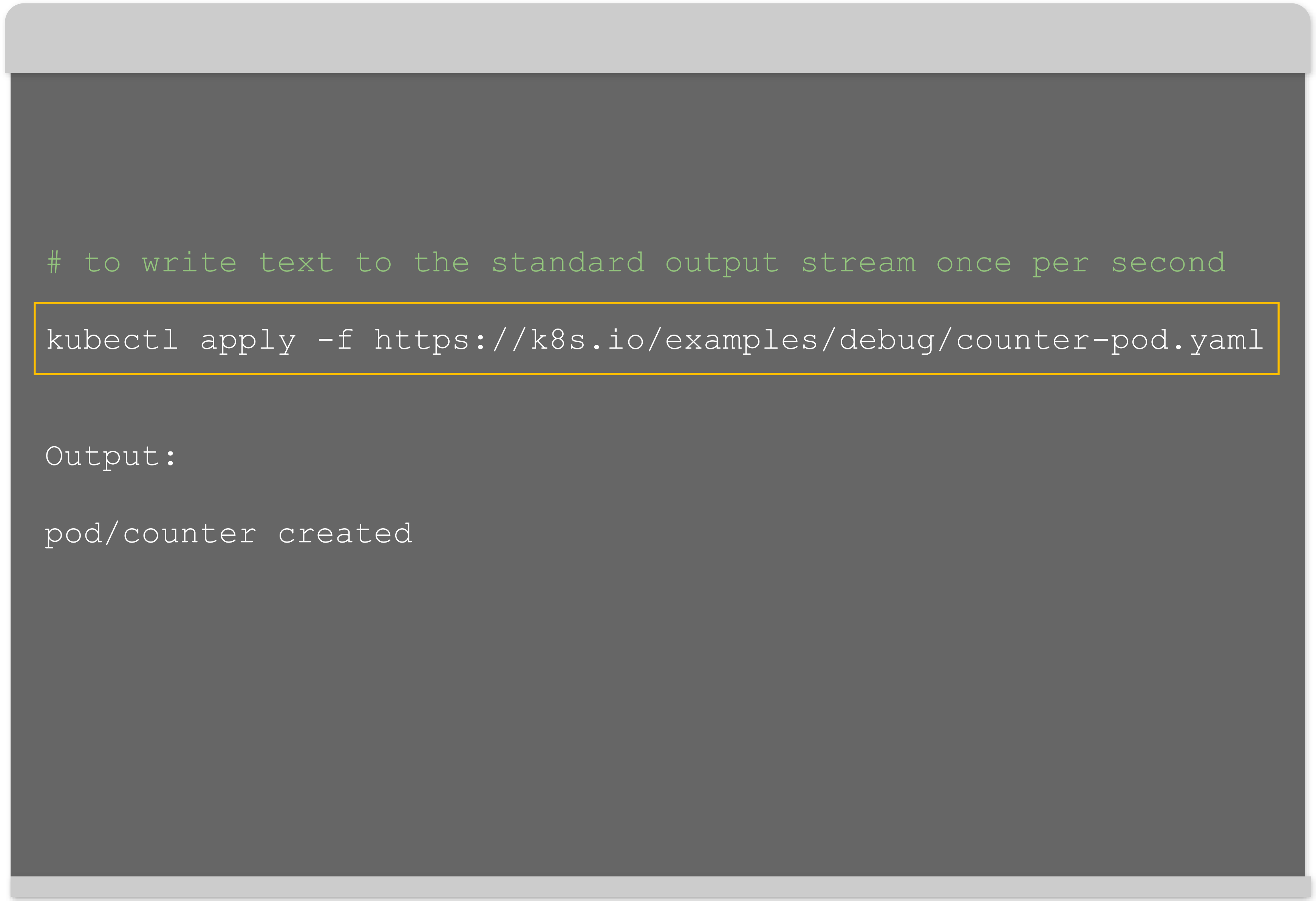
Basic Logging in Kubernetes

The following YAML configuration uses a pod specification with a container to write text to the standard output stream once every second:

```
apiVersion: v1
Kind: pod
Metadata:
   name: counter
spec:
   containers:
 - name:   count
   image:  busybox:1.28
   args : [/bin/sh, -c,
        'i=0; while true; do echo "$i: $(date)"; i=$((i+1));
sleep 1; done']
```

Basic Logging in Kubernetes

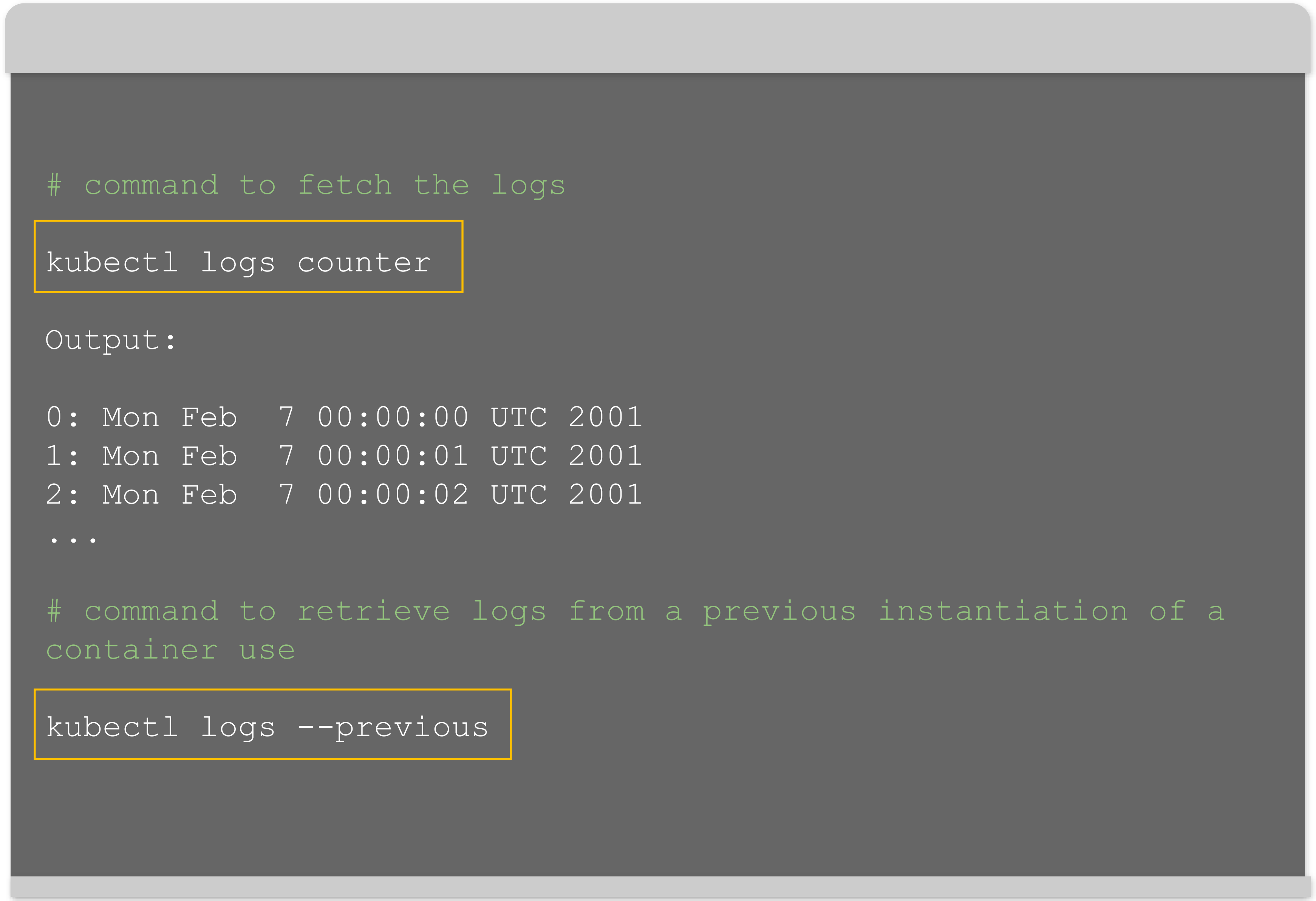
Use the following command to create a pod that outputs text to the standard output stream every second:



```
# to write text to the standard output stream once per second

kubectl apply -f https://k8s.io/examples/debug/counter-pod.yaml

Output:

pod/counter created
```

Fetching Logs in Kubernetes

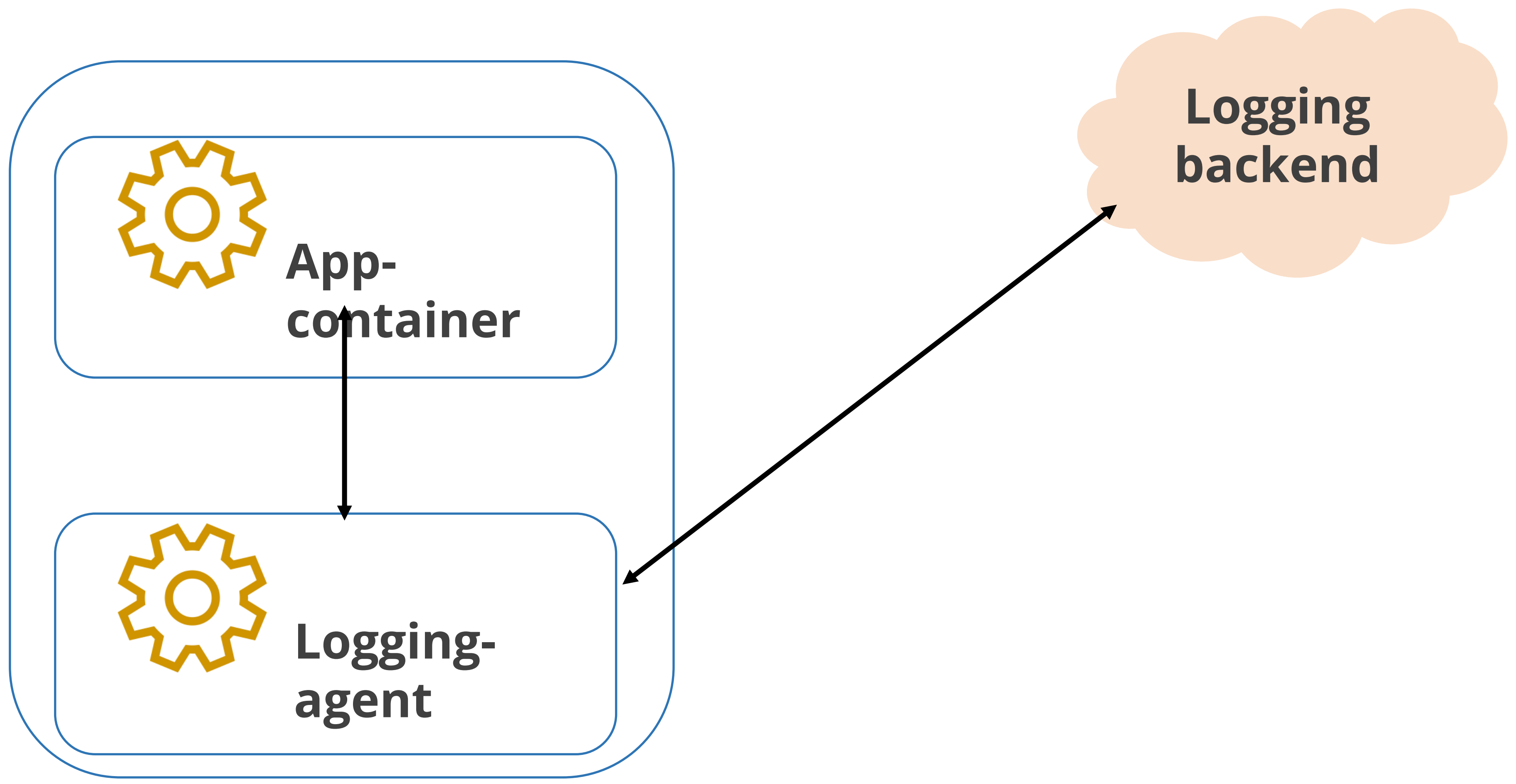
Use the following commands to fetch logs and retrieve logs from a previous container instance:



```
# command to fetch the logs

kubectl logs counter

Output:

0: Mon Feb  7 00:00:00 UTC 2001
1: Mon Feb  7 00:00:01 UTC 2001
2: Mon Feb  7 00:00:02 UTC 2001
...

# command to retrieve logs from a previous instantiation of a
container use

kubectl logs --previous
```

Sidecar Container with a Logging Agent

A sidecar container with a separate logging agent can be configured to run alongside the application container when the node-level logging agent is not flexible enough for specific logging requirements.



In this setup, the logging agent collects logs from the application container and sends them to the logging backend for further analysis and storage.

Configuration Files to Implement Sidecar Container

The following configuration files set up a sidecar container with **Fluentd** as the logging agent, which collects logs from the application container and forwards them to a logging backend.

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: fluentd-config
data:
  fluentd.conf: |
    <source>
      type tail
      format none
      path /var/log/1.log
      pos_file /var/log/1.log.pos
      tag count.format1
    </source>
```

```
<source>
      type tail
      format none
      path /var/log/2.log
      pos_file /var/log/2.log.pos
      tag count.format2
</source>

    <match **>
      type google_cloud
```

Configuration Files to Implement Sidecar Container

The following configuration file defines a pod with a sidecar container running **Fluentd**. This configuration allows **Fluentd** to collect logs from the application and forward them to a specified logging backend.

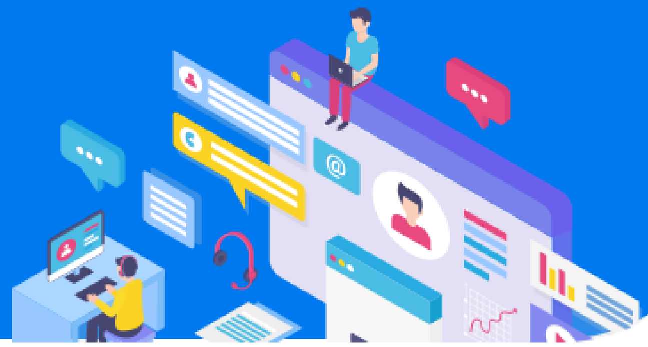
```
apiVersion: v1
kind: Pod
metadata:
  name: counter
spec:
  containers:
  - name: count
    image: busybox:1.28
    args:
    - /bin/sh
    - -c
    - >
i=0;
   while true;
   do
     echo "$i: $(date)" >> /var/log/1.log;
     echo "$(date) INFO $i" >> /var/log/2.log;
     i=$((i+1));
     sleep 1;
   done
```

Configuration Files to Implement Sidecar Container

Fluentd can be replaced with any logging agent, providing flexibility in choosing the right tool for the logging needs.

```
VolumeMounts:
   - name: varlog
     mountPath: /var/log
 - name: count-agent
   image: k8s.gcr.io/fluentd-gcp:1.30
   env:
   - name: FLUENTD_ARGS
     value: -c /etc/fluentd-config/fluentd.conf

VolumeMounts:
   - name: varlog
     mountPath: /var/log
   - name: config-volume
     mountPath: /etc/fluentd-config
 volumes:
 - name: varlog
   emptyDir: {}
 - name: config-volume
   configMap:
     name: fluentd-config
```

Assisted Practice



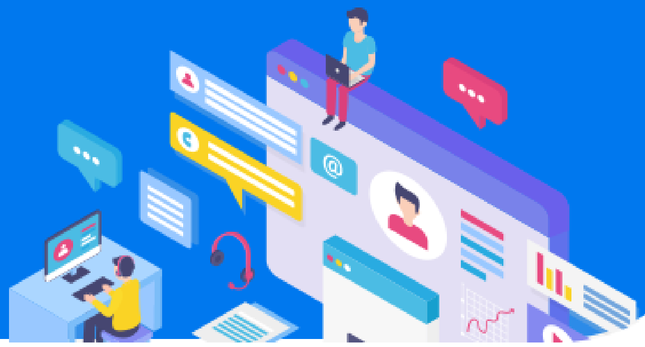
Working with Kubernetes Cluster Logging Architecture **Duration: 10 Min.**

Problem statement:
You have been asked to monitor and manage application logs within Kubernetes clusters by retrieving logs for specific pods and using log options to filter and manage log data.

Outcome:
By the end of this demo, you will be able to monitor and manage application logs within Kubernetes clusters by retrieving logs for specific pods and using log options to filter and manage log data.

Note: Refer to the demo document for detailed steps

Assisted Practice: Guidelines



Steps to be followed:

1. Retrieve log entries for a Kubernetes pod
2. Use the log options and switch information

Quick Check

You are a Kubernetes administrator troubleshooting an issue where a container in one of your pods crashed unexpectedly. You need to investigate the logs from the previous instance of the container to understand what caused the failure.

Which command would you use to retrieve the logs from the previous container instance?

A. kubectl logs

B. kubectl logs --previous

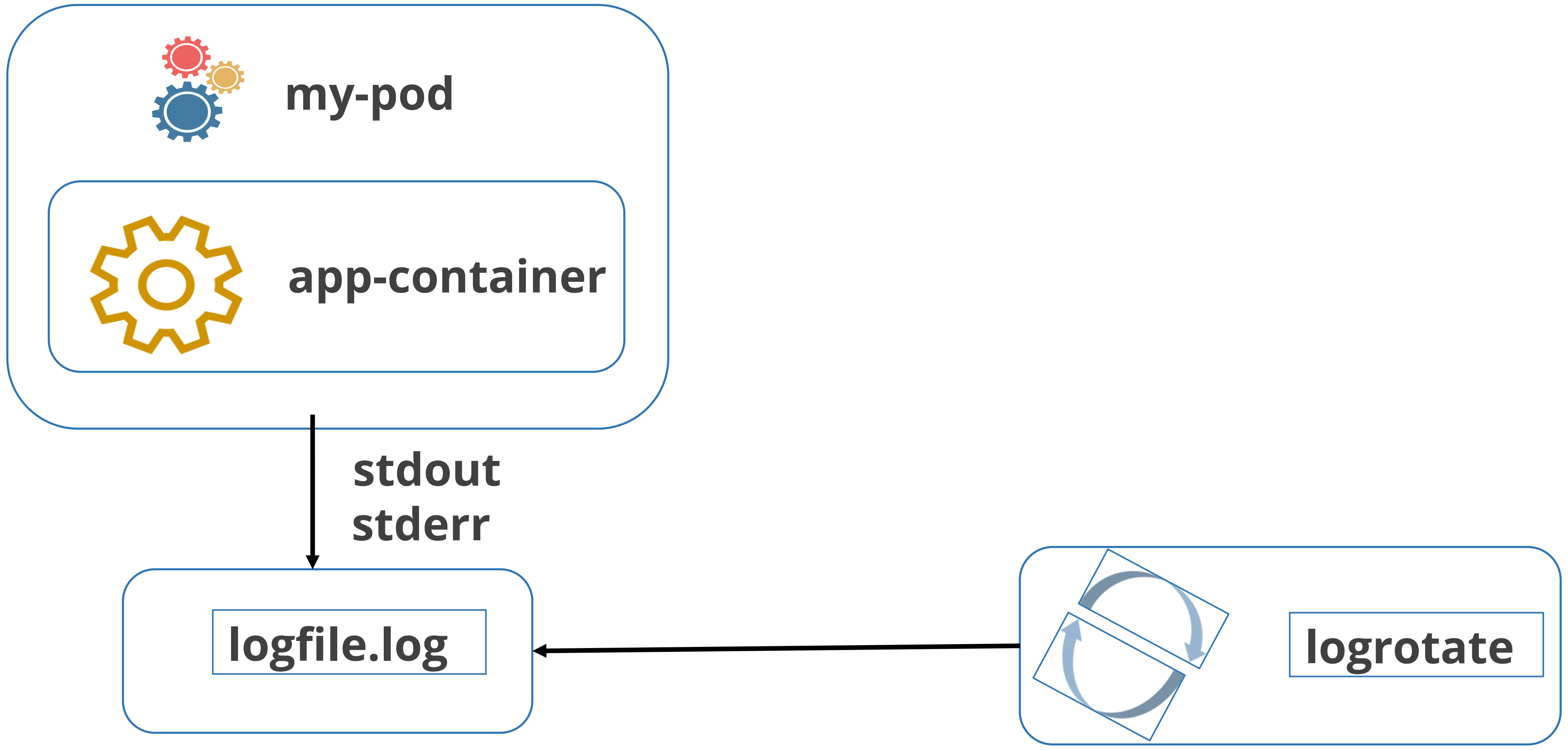
C. kubectl logs counter

D. kubectl logs --since

Cluster and Node Logs

Node-Level Logging

A container engine manages and redirects any output to the application's **stdout** and **stderr** streams. A deployment tool must be set up to manage log rotation.



At the node level, the kubelet and CRI manage logs, using specific flags to control log size and rotation.

CRI Container Management

The kubelet is responsible for managing the log directory structure and handling log rotation when using a CRI container runtime.

There are two kubelet flags that help control log size and file rotation for container logs:

- **Container-log-max-size** to set the maximum size for each log file

- **Container-log-max-files** to set the maximum number of files allowed for each container

In addition to node-level logging, Kubernetes system components generate logs, depending on whether they run inside or outside containers.

System Component Logs

There are two types of system components in Kubernetes that generate logs:

Those that run a container

Example: Kubernetes Scheduler and kube-proxy

Those that do not run a container

Example: Kubelet and Container runtime

Cluster-Level Logging

Cluster-level logging centralizes log data from containers, nodes, and pods across the entire Kubernetes cluster.

Some methods that can be considered for cluster-level logging include:

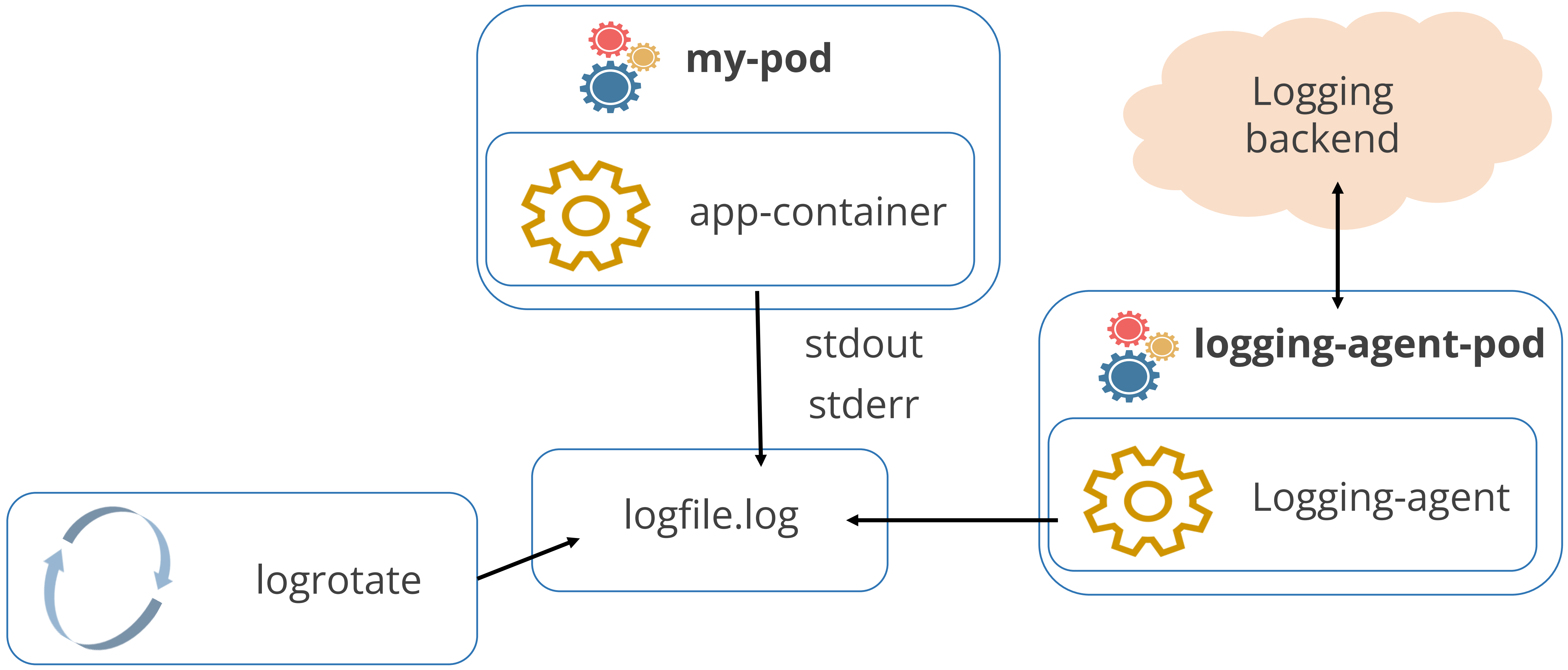
Using a node-level logging agent that runs on every node

Using a dedicated sidecar container for logging in an application pod

Pushing logs directly to a backend from an application

Usage of Node Logging Agent

Cluster-level logging may be implemented by including a node-level logging agent (a tool that pushes logs to a backend) on each node.



Another method involves using sidecar containers for logging.

Usage of Sidecar Container

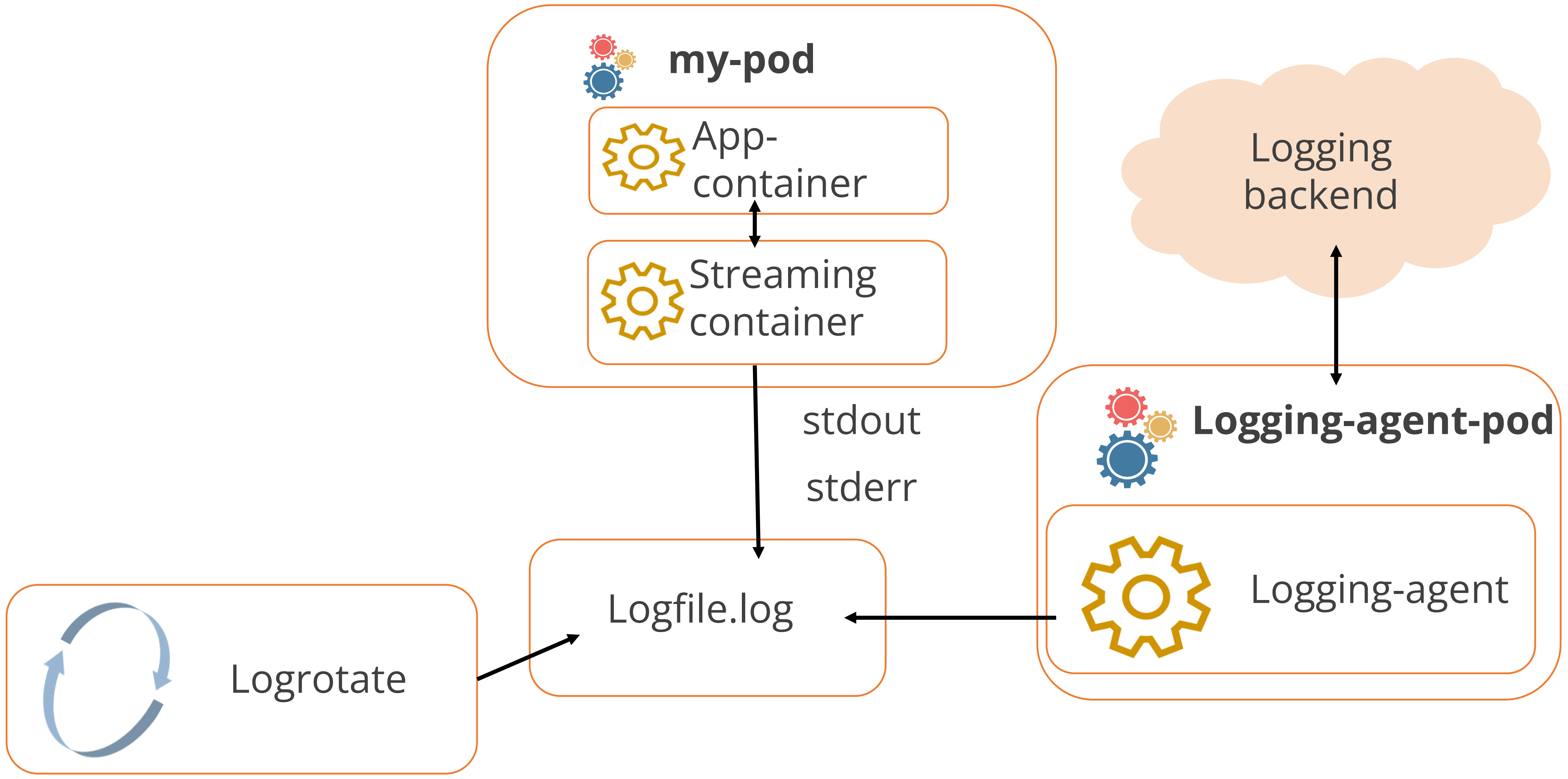
A sidecar container can be used for cluster-level logging in either of the following ways:

By streaming application logs to its own stdout

By running a logging agent that is configured to pick up logs from an application container

Usage of Sidecar Container with a Logging Agent

A sidecar container prints logs to its **stdout** or **stderr** stream, which can then be collected by a logging agent for further processing and forwarding.



Pod with a Single Sidecar Container

Here is a configuration file for a pod with a single sidecar container that collects log data:

```
apiVersion: v1
kind: Pod
metadata:
  name: counter
spec:
  containers:
  - name: count
    image: busybox:1.28
    args:
    - /bin/sh
    - -c
    - >
i=0;
   while true;
   do
      echo "$i: $(date)" >> /var/log/1.log;
      echo "$(date) INFO $i" >> /var/log/2.log;
      i=$((i+1));
      sleep 1;
   done
```

```
volumeMounts:
    - name: varlog
      mountPath: /var/log
  volumes:
  - name: varlog
    emptyDir: {}
```

Pod with Two Sidecar Containers

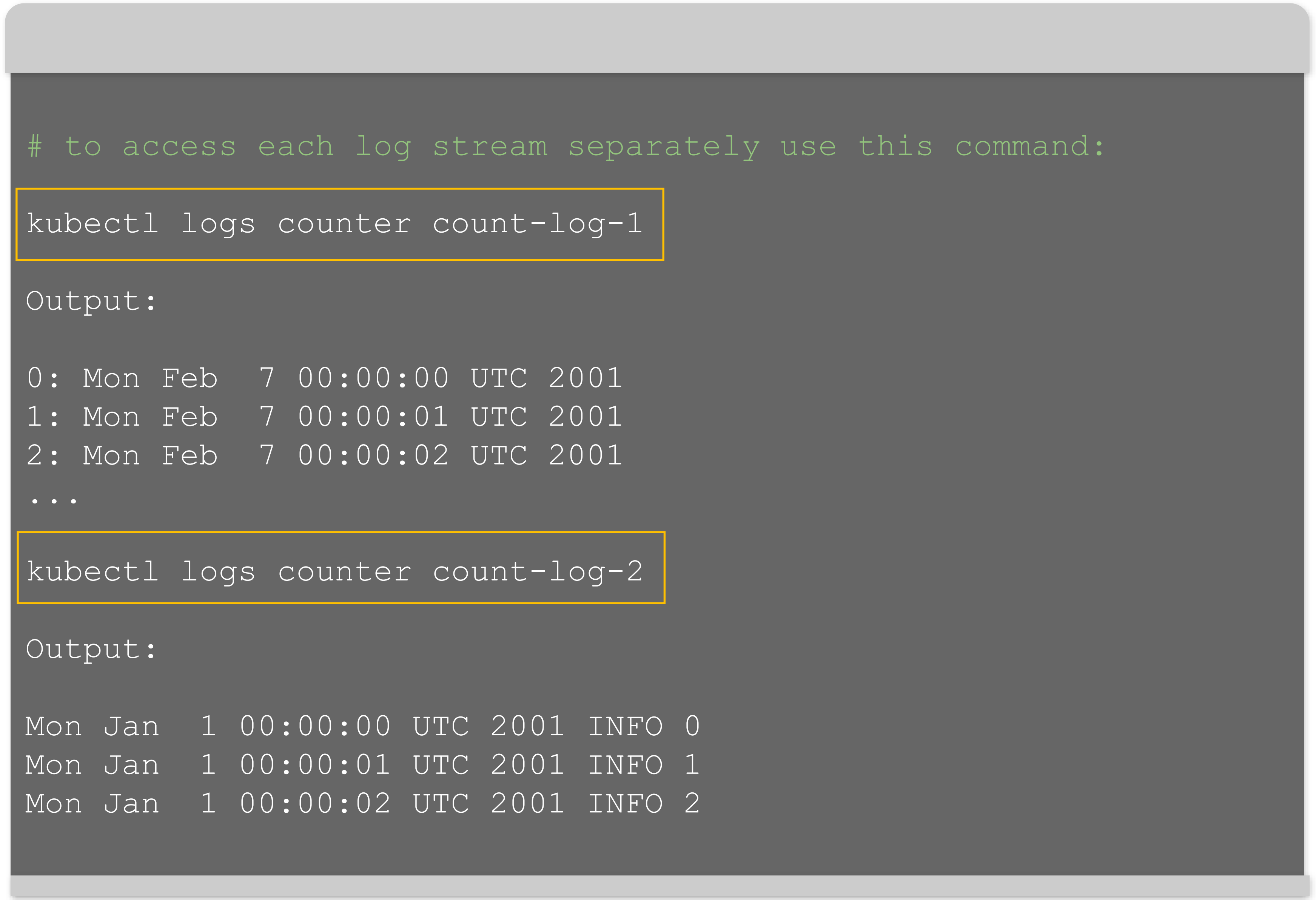
Here is a configuration file for a pod with two sidecar containers, each handling log data:

```
apiVersion: v1
kind: Pod
metadata:
  name: counter
spec:
  containers:
  - name: count
    image: busybox:1.28
    args:
    - /bin/sh
    - -c
    - >
i=0;
   while true;
   do
      echo "$i: $(date)" >> /var/log/1.log;
      echo "$(date) INFO $i" >> /var/log/2.log;
      i=$((i+1));
      sleep 1;
   done
```

```
volumeMounts:
    - name: varlog
      mountPath: /var/log
  - name: count-log-1
    image: busybox:1.28
    args: [/bin/sh, -c, 'tail -n+1 -f
/var/log/1.log’]
volumeMounts:
    - name: varlog
      mountPath: /var/log
  - name: count-log-2
    image: busybox:1.28
    args: [/bin/sh, -c, 'tail -n+1 -f
/var/log/2.log']
    volumeMounts:
    - name: varlog
      mountPath: /var/log
  volumes:
  - name: varlog
    emptyDir: {}
```

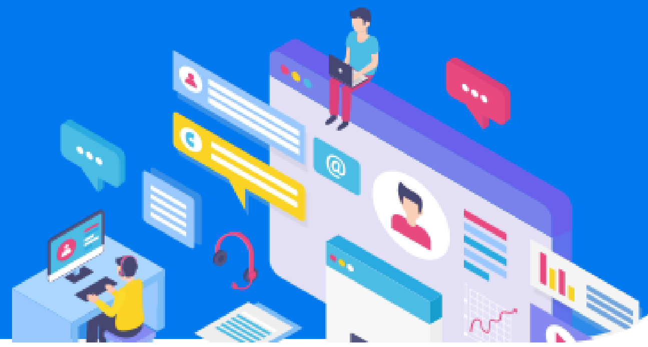
Access Log Streams

When the pod is run, each log stream can be accessed separately using the following command:



```
# to access each log stream separately use this command:

kubectl logs counter count-log-1

Output:

0: Mon Feb  7 00:00:00 UTC 2001
1: Mon Feb  7 00:00:01 UTC 2001
2: Mon Feb  7 00:00:02 UTC 2001
...

kubectl logs counter count-log-2

Output:

Mon Jan  1 00:00:00 UTC 2001 INFO 0
Mon Jan  1 00:00:01 UTC 2001 INFO 1
Mon Jan  1 00:00:02 UTC 2001 INFO 2
```

Assisted Practice



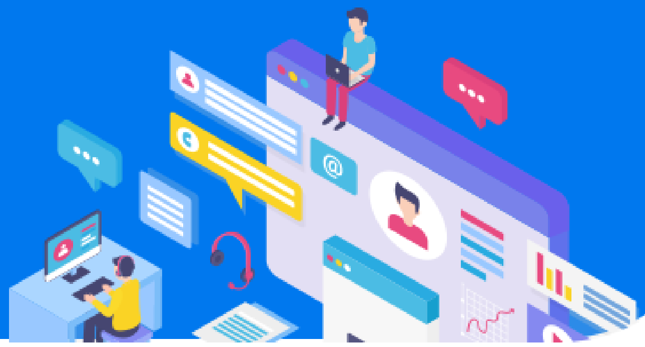
Inspecting Cluster and Node Logs **Duration: 10 Min.**

Problem statement:
You have been asked to inspect the control-plane components like the API server, controller manager, etcd, and kubelet service in worker nodes for monitoring and troubleshooting.

Outcome:
By the end of this assisted practice, you will be able to inspect the control-plane components like the API server, controller manager, etcd, and kubelet service in worker nodes for effective monitoring and troubleshooting.

Note: Refer to the demo document for detailed steps

Assisted Practice: Guidelines



Steps to be followed:

1. View control-plane component logs
2. View the controller manager logs
3. View the etcd logs
4. View worker node logs

Assisted Practice

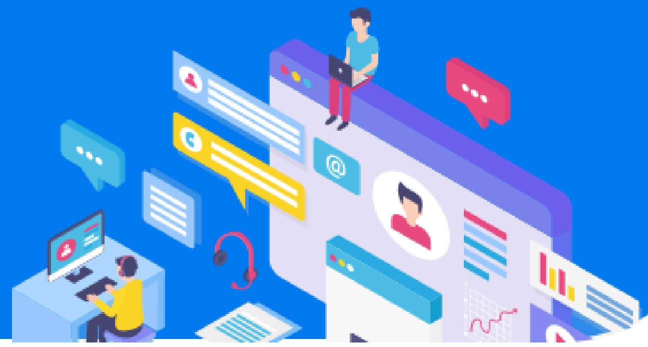
Troubleshooting Node Readiness **Duration: 10 Min.**

Problem statement:
You have been asked to diagnose and troubleshoot the issue of a worker node transitioning from **Not Ready** to **Ready** status.

Outcome:
By the end of this demo, you will be able to diagnose and troubleshoot issues related to worker nodes transitioning from **NotReady** to **Ready** status.

Note: Refer to the demo document for detailed steps

Assisted Practice: Guidelines



Steps to be followed:

1. Check the node status on the master node
2. Disable the worker-node-2 and troubleshoot the issue
3. Fix the worker-node-2

Quick Check

You are working as a Kubernetes administrator and have set up multiple containers within a pod that are generating separate log streams. One of your developers reports that only specific log streams need to be checked for debugging purposes. You decide to access the logs of each container separately to analyze the issue.

Which command should you use to access the log stream of a specific container in the pod?

A. kubectl logs counter

B. kubectl logs counter --previous

C. kubectl logs --all-containers=true

D. kubectl logs counter count-log-1

Container Logs

Fetch Container Logs

Fetching container logs is essential for diagnosing issues that occur at the container level in a Kubernetes cluster.

Kubectl provides the following command to fetch the logs:

```
kubectl logs counter
```

If there are many containers in a single pod, the names of the containers should be specified in the command.

Commands to Fetch Container Logs

The logs of a container can be fetched using any of the following options:

kubectl logs nginx	To get the snapshot logs from pod nginx with only one container
kubectl logs –p –c ruby web-1	To get a snapshot of ruby container logs that were terminated earlier

Commands to Fetch Container Logs

kubectl logs –f –c ruby web-1	To start streaming the logs of the ruby container in a pod called web-1
kubectl logs –-tail=20 nginx	To show the most recent 20 lines of output in pod nginx
kubectl logs –since=1h nginx	To display all the logs from the pod nginx written in the last one hour

Fetch Containerd Container Logs

Retrieve logs from containers managed by the containerd runtime using the crictl command.

This command gathers logs present at the time of execution:

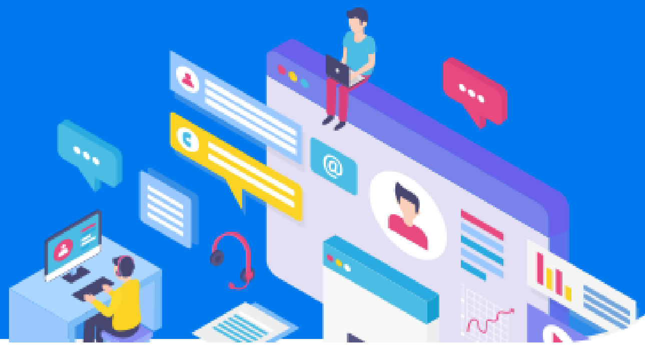
```
$ crictl logs [OPTIONS] CONTAINER
```

This command is useful in environments where docker is not used as the runtime, providing flexibility for monitoring and troubleshooting container logs.

Docker Container Command Options

Name	Description
--details	Displays additional information provided in the logs
--follow, --f	Continuously streams log output as it is updated
--since	Shows logs starting from a specified timestamp
--tail <number>	Specifies how many lines to display starting from the most recent logs
--until	Displays logs up to a specified timestamp

Assisted Practice



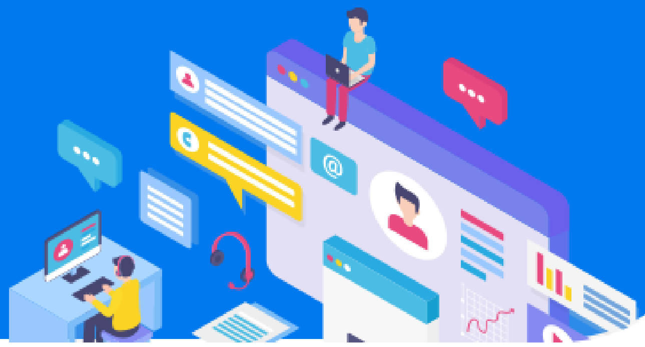
Understanding Container Logs **Duration: 10 Min.**

Problem statement:
You have been asked to view and check the container logs within a Kubernetes cluster using the crictl commands for monitoring runtime operations.

Outcome:
By the end of this demo, you will be able to view and check container logs within a Kubernetes cluster using the crictl commands for monitoring runtime operations.

Note: Refer to the demo document for detailed steps

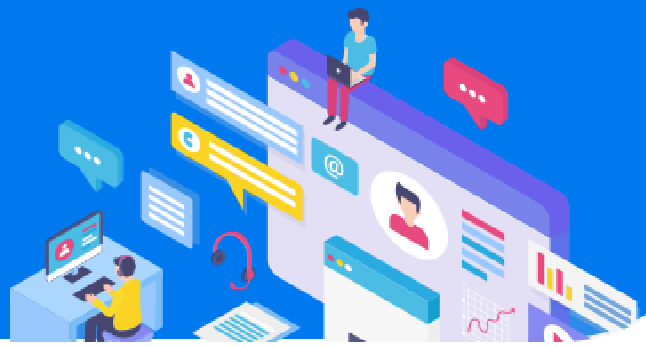
Assisted Practice: Guidelines



Steps to be followed:

1. Check the container logs using the crictl commands

Assisted Practice



Analyzing Pod Logs

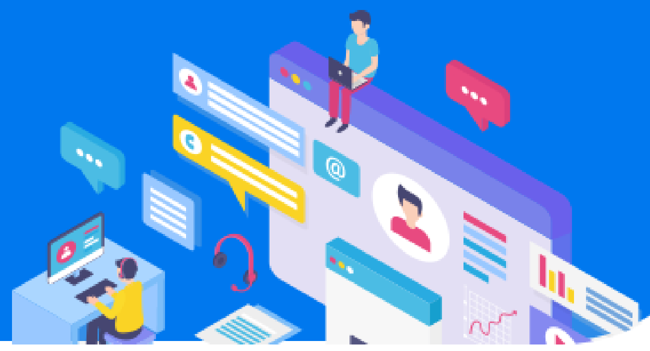
Duration: 10 Min.

Problem statement:
You have been assigned a task to analyze the logs of a pod hosting a container within a Kubernetes cluster to view the application or service's output.

Outcome:
By the end of this demo, you will be able to analyze the logs of a pod hosting a container within a Kubernetes cluster.

Note: Refer to the demo document for detailed steps

Assisted Practice: Guidelines



Steps to be followed:

1. Configure and verify Nginx deployment

Quick Check

You are investigating an issue with the nginx pod in your Kubernetes cluster. The developer asks you to check the logs from the last hour to identify any potential issues.
Which of the following commands will help you retrieve these logs?

A. kubectl logs --tail=20 nginx

B. kubectl logs -f nginx

C. kubectl logs --since=1h nginx

D. kubectl logs --previous nginx

Application Troubleshooting

Diagnose the Problem

The first step in diagnosing a problem in the application is to identify where the problem is located. It could be in any of the following components:

- Pods
- Services
- ReplicaSets

Debug Pods

The first step in debugging a pod is to check its current state and recent events with the following command:

```
# to check the current state of the Pod and recent events

kubectl describe pods ${POD_NAME}
```

Pods in Pending State

When a pod gets stuck in a pending state, it indicates its inability to be scheduled onto a node. There could be two reasons for this:

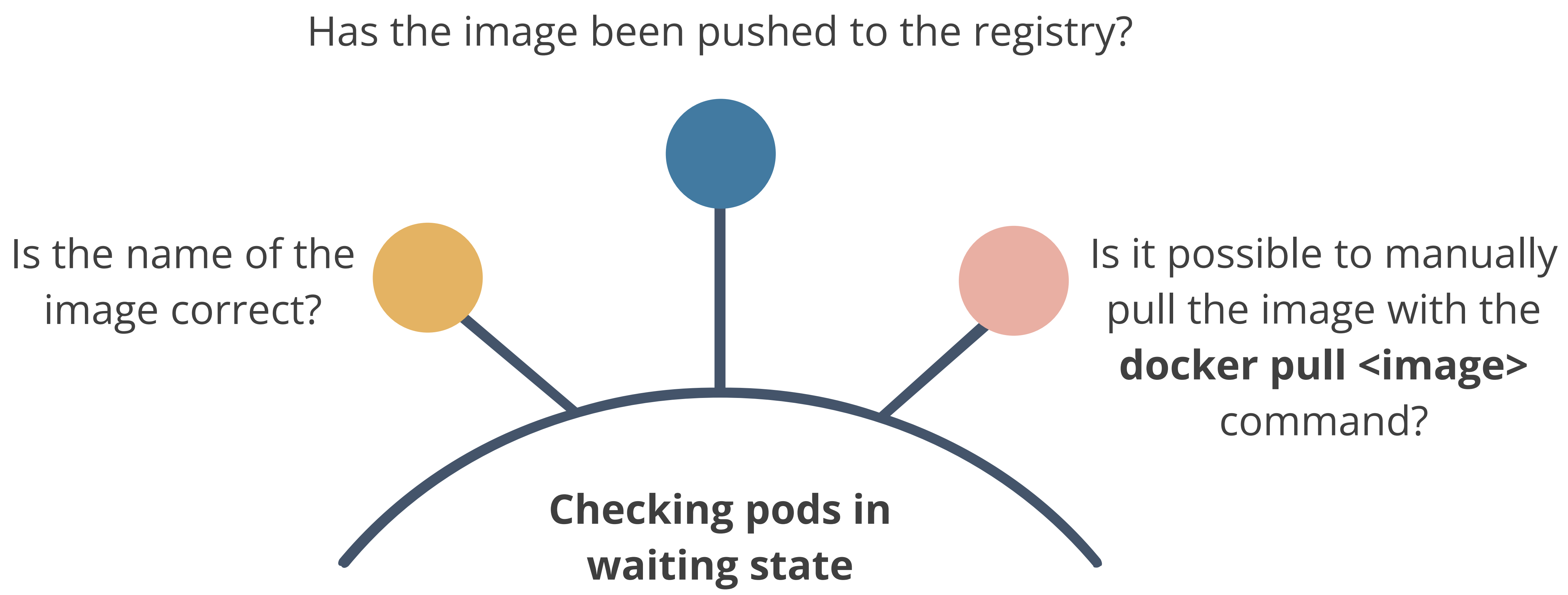
1. **Not enough resources:** The supply of CPU and memory in the cluster is depleted.

2. **Pod bound to a hostPort:** In this case, there are only a few places where the pod can be scheduled.

Pods in Waiting State

Failure to pull the image is a common cause of waiting pods. When troubleshooting pods in a waiting state, it is essential to verify if the image-related issues are causing delays.

Use the following steps to investigate the root cause:


Has the image been pushed to the registry?
Is the name of the image correct?
Is it possible to manually pull the image with the **docker pull <image>** command?
Checking pods in waiting state

Pods Not Behaving as Expected

If a pod is not functioning as expected, it could be due to configuration issues.

Common reasons for pod misbehavior include:

- An error in the pod description was ignored during the pod creation
- A section of the pod description is nested incorrectly, or a key name is typed incorrectly

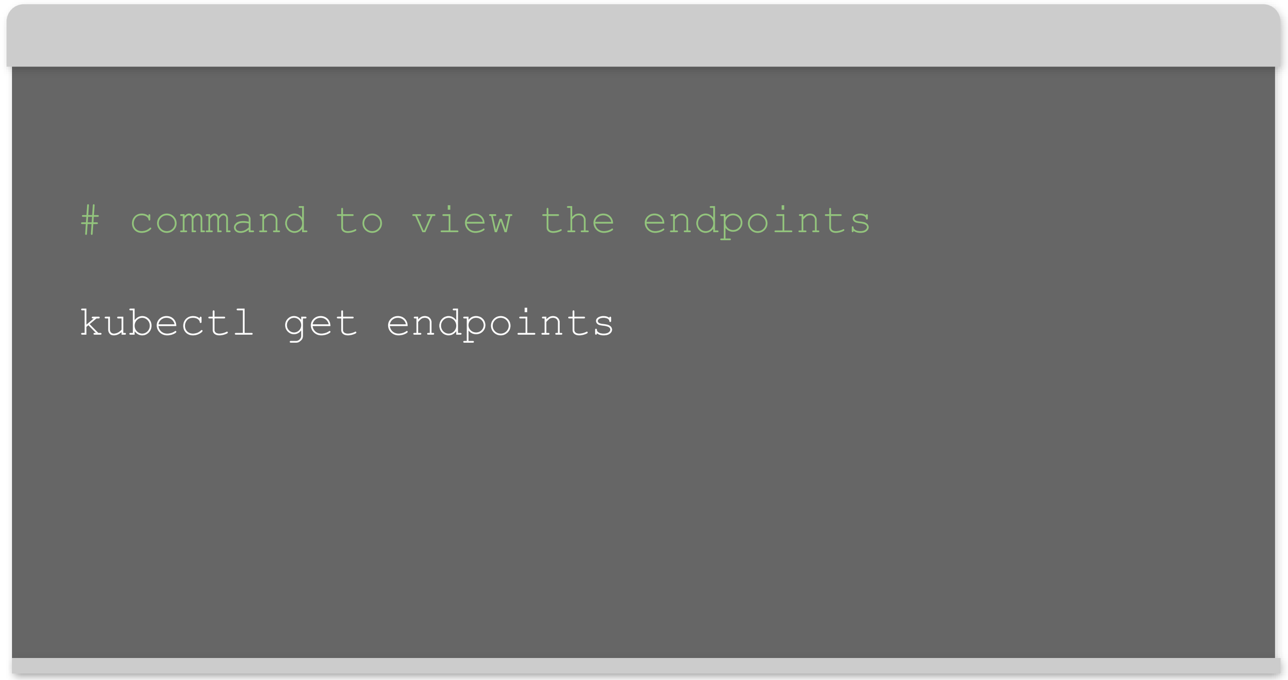
Debugging the Services

Services provide load balancing across a set of pods. When issues arise, they can be debugged by performing the following checks:

- Check whether endpoints are available for the service
- Ensure that endpoints match the number of pods that are expected to be members of the service
- List pods using labels that the service uses

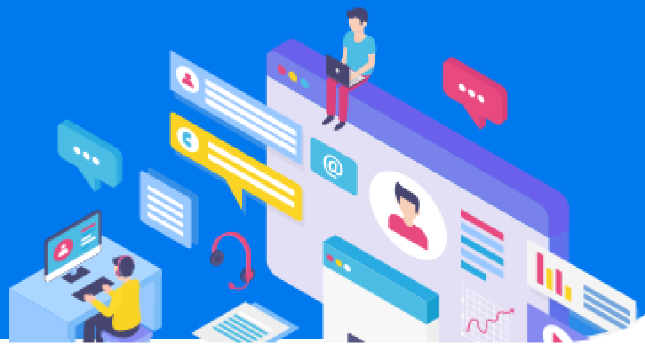
Command for Debugging Services

The number of endpoints for a service should match the number of pods available for that service. Use the following command to check if the service endpoints are properly configured and available:



```
# command to view the endpoints

kubectl get endpoints
```

Assisted Practice



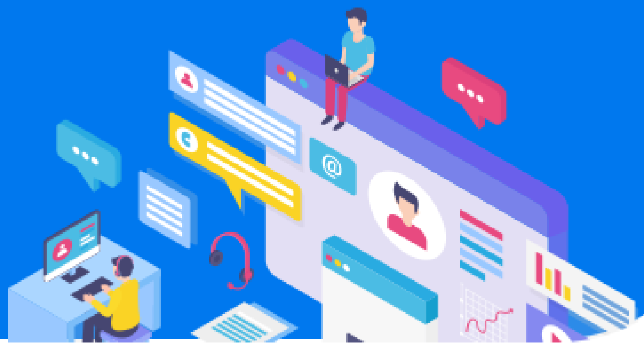
Troubleshooting an Application Pod in Kubernetes **Duration: 10 Min.**

Problem statement:
You have been asked to set up an application pod in Kubernetes, diagnose potential issues, and implement necessary troubleshooting steps to ensure successful application deployment.

Outcome:
By the end of this demo, you will be able to set up an application pod in Kubernetes, diagnose potential issues, and implement necessary troubleshooting steps to ensure successful application deployment.

Note: Refer to the demo document for detailed steps

Assisted Practice: Guidelines



Steps to be followed:

1. Set up and diagnose the application pod

Quick Check

You are troubleshooting a pod in your Kubernetes cluster that is not running as expected. Before diving into specific logs, you want to check their current state and any recent events that might have occurred. Which command will help you retrieve the pod's status and recent events?

A. kubectl get pods

B. kubectl describe pods ${POD_NAME}

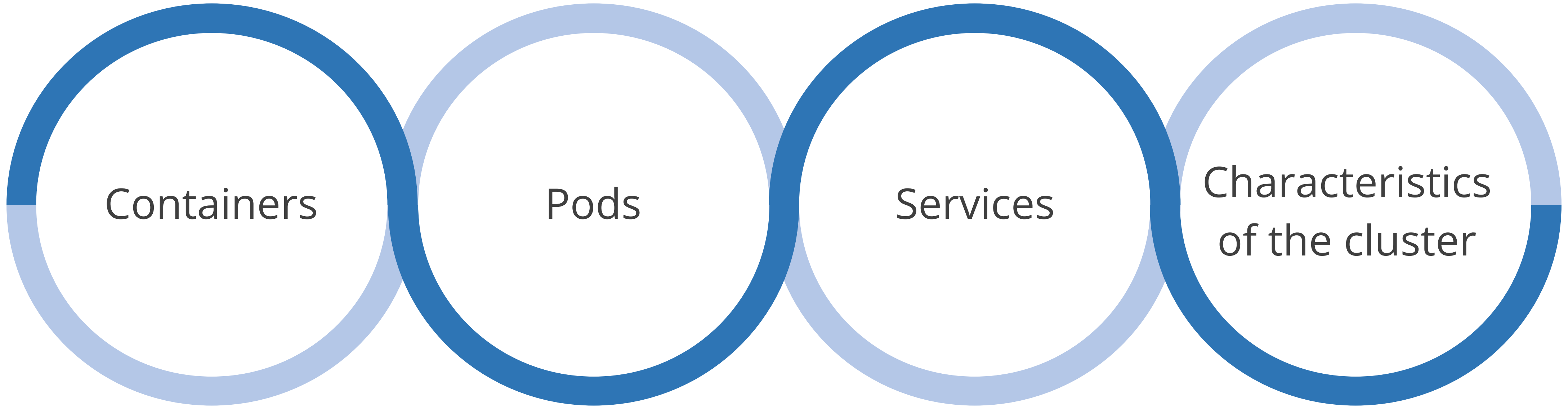
C. kubectl logs ${POD_NAME}

D. kubectl delete pods ${POD_NAME}

Monitoring Tools

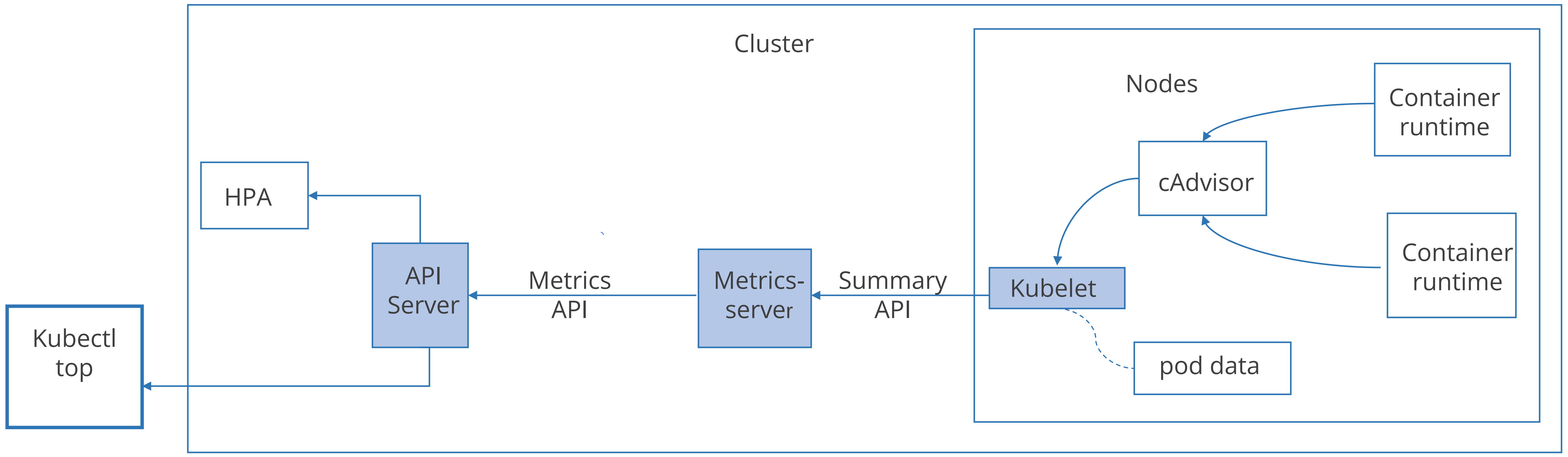
Elements for Monitoring Resources

Kubernetes offers detailed insights into the resource usage of applications. Effective monitoring of application performance in a Kubernetes cluster can be done by evaluating the following components:



Architecture of Resource Metrics Pipeline

The Kubernetes metrics API provides essential metrics for auto-scaling and related use cases. The diagram below shows how Kubernetes collects, processes, and serves metrics for scaling decisions:



Architecture of Resource Metrics Pipeline

Each component plays a critical role in gathering, processing, and exposing metrics for monitoring and scaling workloads in Kubernetes environments.

cAdvisor

Kubelet includes a daemon for collecting, aggregating, and exposing container metrics.

kubelet

The kubelet acts as a node agent responsible for managing container resources. It provides access to resource metrics via the /metrics/resource and /stats API endpoints.

Summary API

The kubelet provides an API for discovering and retrieving per-node summarized stats via the /stats endpoint.

Architecture of Resource Metrics Pipeline

Metrics-server

The Metrics Server is an add-on for Kubernetes clusters that gathers and aggregates node and pod resource usage data. It is served by the API server for use by HPA, VPA, and the kubectl top command.

Metrics API

Access to CPU and memory for workload autoscaling is supported by the Kubernetes API. There will be a need for an API extension server that supports the metrics API to make this work in the cluster.

Resource Metrics Pipeline

The resource metrics pipeline only provides a small set of metrics for cluster components like the **Horizontal Pod Autoscaler (HPA)** controller and the **kubectl top** function.

The metrics server discovers the nodes in the cluster and queries each node’s kubelet to get the memory and CPU usage.

The kubelet is like a bridge between the master and the nodes. It translates each pod into its containers and fetches container usage statistics.

The kubelet displays the pod resource usage statistics through the metrics server API.

Full Metrics Pipeline

Kubernetes responds to metrics by scaling or adapting the cluster based on its current state.

The monitoring pipeline collects the metrics from the kubelet and exposes them via an adapter through the following APIs:

custom.metrics.k8s.io

external.metrics.k8s.io

These APIs allow Kubernetes to access metrics beyond standard CPU and memory usage, supporting custom scaling policies and external metric integration.

Metrics API

The Metrics API is a critical component that provides real-time resource usage data, enabling Kubernetes to make informed decisions about scaling and optimizing resource utilization.

1

Is discoverable through the same endpoint as other Kubernetes APIs under the path **/apis/metrics.K8s.Io/**

2

Offers security, reliability, and scalability

Measure Resource Usage

The following are the resource usage parameters that are monitored while troubleshooting:

CPU

CPU denotes compute processing and is reported as the average use in CPU cores over a period.

Memory

The working set is the portion of memory currently in use that cannot be released. While capturing memory metrics, the amount of active memory at a specific moment is captured.

Metrics Server

The metrics server is the cluster-wide aggregator of resource usage data. It collects metrics from the summary API.

Quick Check

You are responsible for monitoring resource usage in your Kubernetes cluster to ensure efficient scaling. As part of this task, you need to use an API that provides real-time resource metrics for pods and nodes. You also want to ensure that the API you use is secure, reliable, and scalable.
Which API would you use, and where can it be accessed?

A. The Metrics API, accessible under the path **/apis/metrics.k8s.io/**

B. The Resource API, accessible under the path **/apis/resource.k8s.io/**

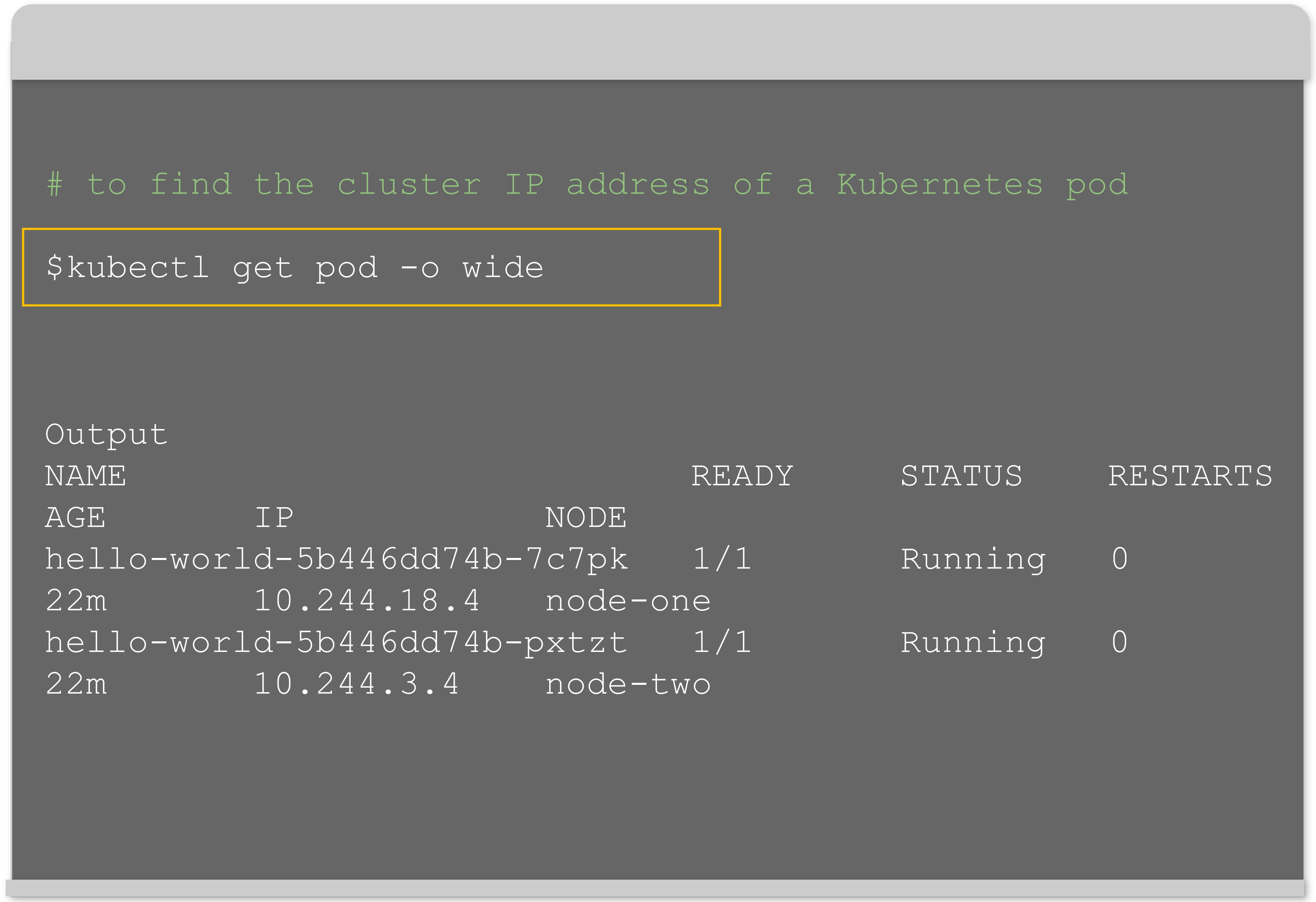
C. The Cluster API, accessible under the path **/apis/cluster.k8s.io/**

D. The Pod Health API, accessible under the path **/apis/health.k8s.io/**

Commands to Debug Networking Issues

Finding a Pod's Cluster IP

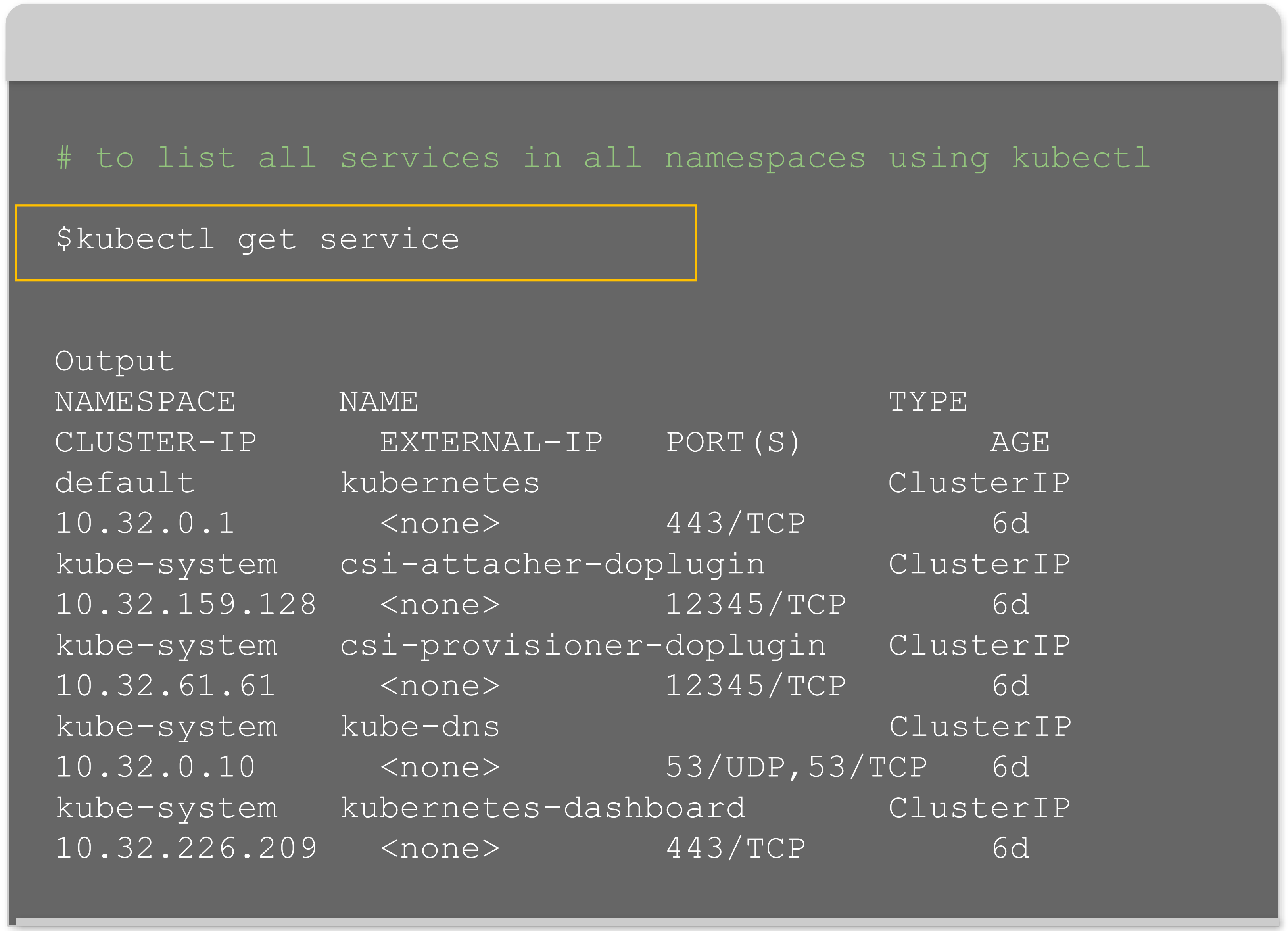
Run the following command to find the Cluster IP address of a Kubernetes pod:



```
# to find the cluster IP address of a Kubernetes pod

$kubectl get pod -o wide



Output
NAME                             READY     STATUS    RESTARTS
AGE        IP            NODE
hello-world-5b446dd74b-7c7pk    1/1       Running    0
22m        10.244.18.4   node-one
hello-world-5b446dd74b-pxtzt    1/1       Running    0
22m        10.244.3.4    node-two
```

Finding the Service IP

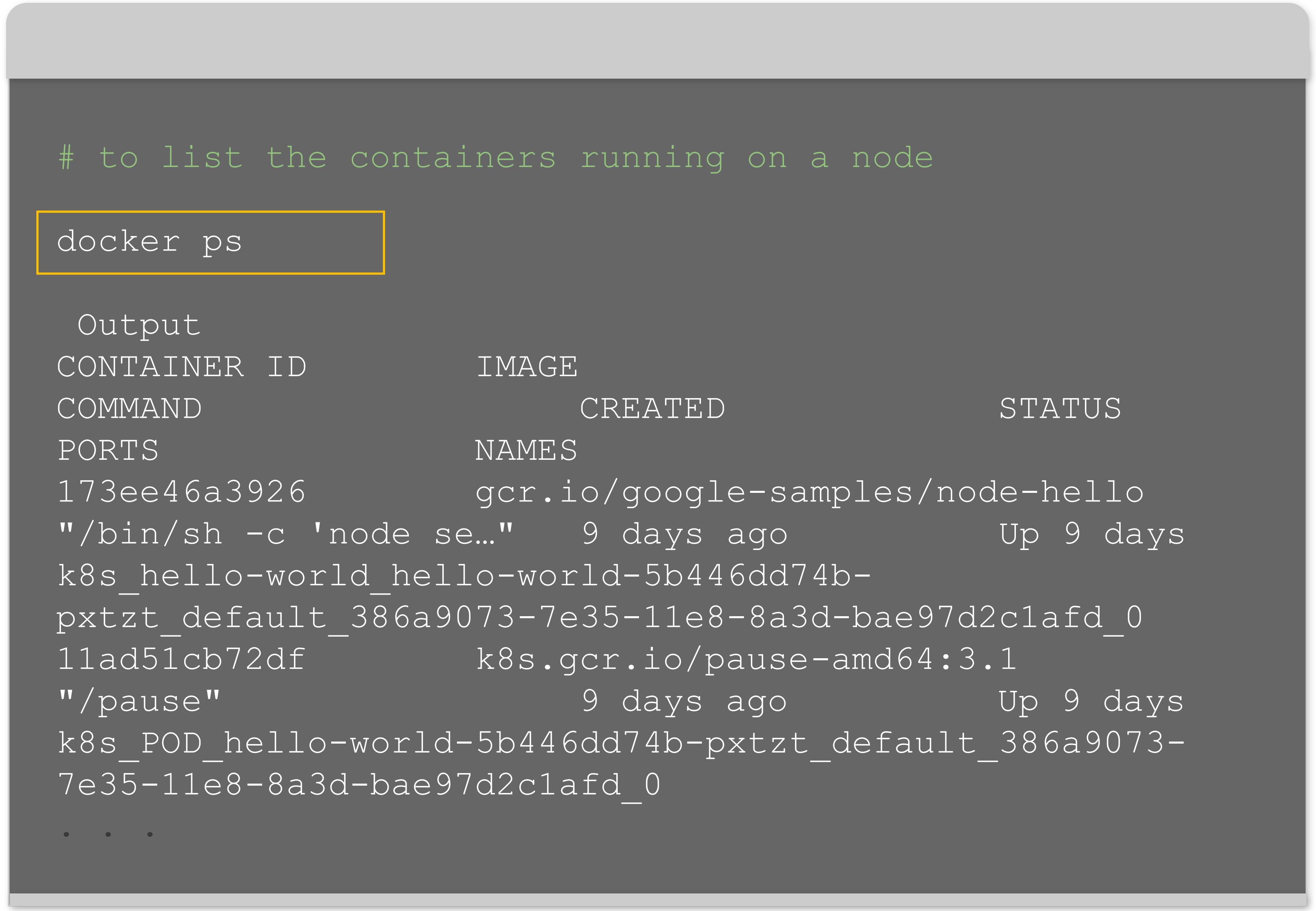
Run the following command to find the service IP addresses under the CLUSTER-IP column:



```
# to list all services in all namespaces using kubectl

$kubectl get service


Output
NAMESPACE     NAME                           TYPE
CLUSTER-IP      EXTERNAL-IP    PORT(S)          AGE
default       kubernetes                     ClusterIP
10.32.0.1       <none>         443/TCP          6d
kube-system   csi-attacher-doplugin          ClusterIP
10.32.159.128   <none>         12345/TCP        6d
kube-system   csi-provisioner-doplugin       ClusterIP
10.32.61.61     <none>         12345/TCP        6d
kube-system   kube-dns                       ClusterIP
10.32.0.10      <none>         53/UDP,53/TCP    6d
kube-system   kubernetes-dashboard           ClusterIP
10.32.226.209   <none>         443/TCP          6d
```

Find and Enter Pod Network Namespaces

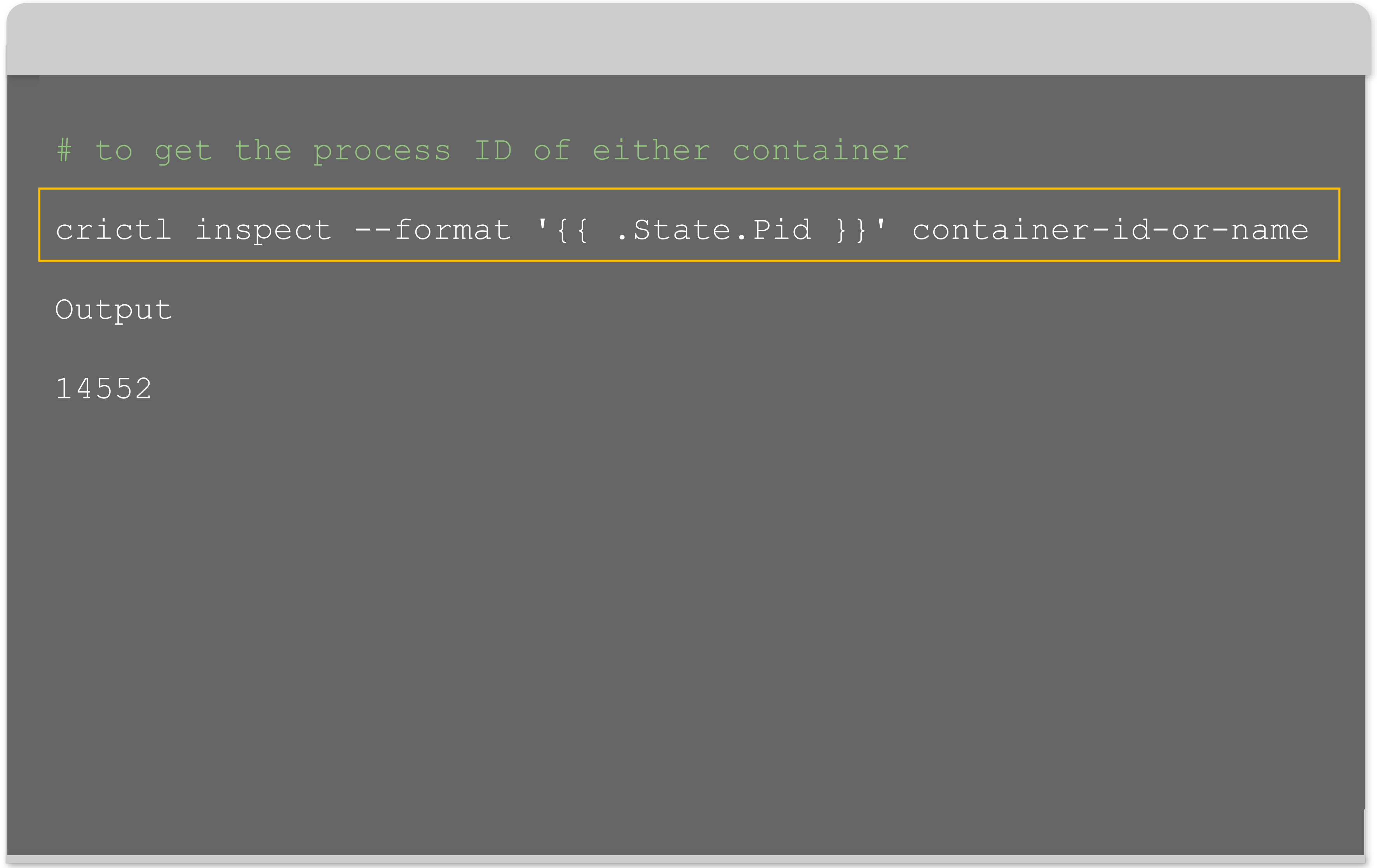
List all the containers running on a node using the following command:



```
# to list the containers running on a node

docker ps

 Output
CONTAINER ID        IMAGE
COMMAND                  CREATED             STATUS
PORTS               NAMES
173ee46a3926        gcr.io/google-samples/node-hello
"/bin/sh -c 'node se…"   9 days ago          Up 9 days
k8s_hello-world_hello-world-5b446dd74b-
pxtzt_default_386a9073-7e35-11e8-8a3d-bae97d2c1afd_0
11ad51cb72df        k8s.gcr.io/pause-amd64:3.1
"/pause"                 9 days ago          Up 9 days
k8s_POD_hello-world-5b446dd74b-pxtzt_default_386a9073-
7e35-11e8-8a3d-bae97d2c1afd_0
. . .
```

Find and Enter Pod Network Namespaces

Use the following command to find out the process ID of the container in the pod that needs to be examined:



```
# to get the process ID of either container

crictl inspect --format '{{ .State.Pid }}' container-id-or-name

Output

14552
```

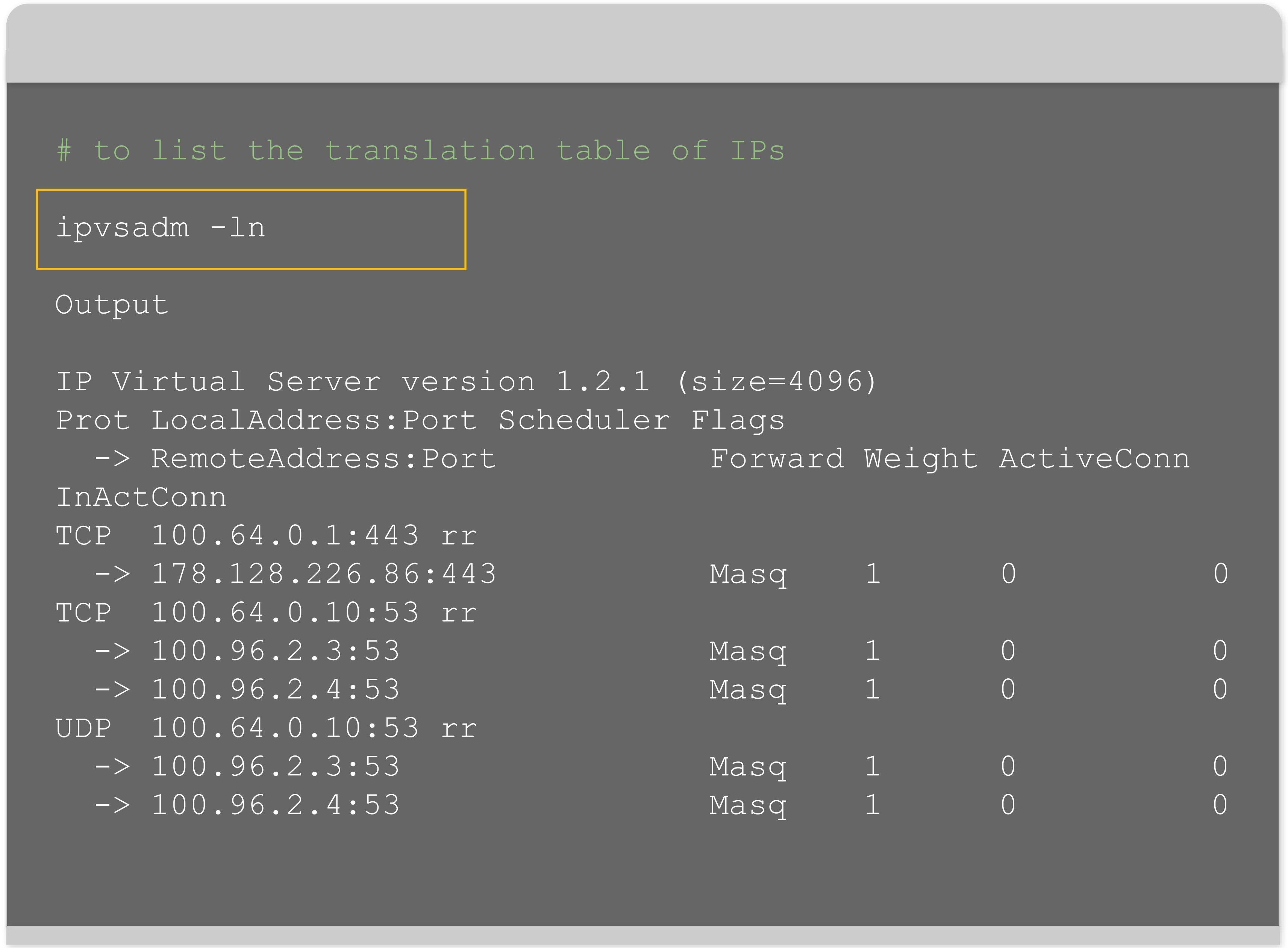
Inspect the IP Table Rules

Use the following **IP table** command to inspect the IP table rules:

```
# to dump all iptables rules on a node

iptables-save

# to list just the Kubernetes Service NAT rules

iptables -t nat -L KUBE-SERVICES

Output

Chain KUBE-SERVICES (2 references)
target     prot opt source               destination
KUBE-SVC-TCOU7JCQXEZGVUNU  udp  --  anywhere
10.32.0.10           /* kube-system/kube-dns:dns cluster IP */
udp dpt:domain
KUBE-SVC-ERIFXISQEP7F7OF4  tcp  --  anywhere
10.32.0.10           /* kube-system/kube-dns:dns-tcp cluster IP
*/ tcp dpt:domain
KUBE-SVC-XGLOHA7QRQ3V22RZ  tcp  --  anywhere
10.32.226.209        /* kube-system/kubernetes-dashboard:
cluster IP */ tcp dpt:https
. . .
```

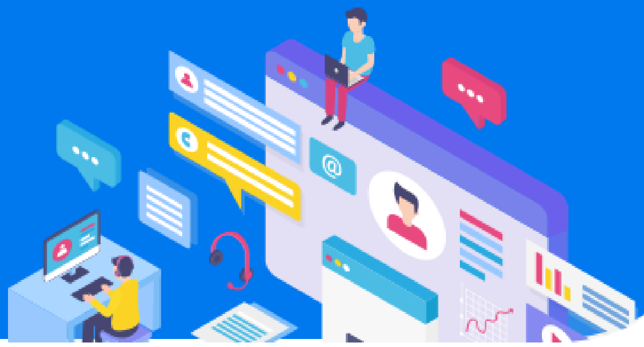
Examine IPVS Details

Use the following command to list the translation table of IPs:



```
# to list the translation table of IPs

ipvsadm -ln

Output

IP Virtual Server version 1.2.1 (size=4096)
Prot LocalAddress:Port Scheduler Flags
  -> RemoteAddress:Port           Forward Weight ActiveConn
InActConn
TCP  100.64.0.1:443 rr
  -> 178.128.226.86:443           Masq    1      0          0
TCP  100.64.0.10:53 rr
  -> 100.96.2.3:53                Masq    1      0          0
  -> 100.96.2.4:53                Masq    1      0          0
UDP  100.64.0.10:53 rr
  -> 100.96.2.3:53                Masq    1      0          0
  -> 100.96.2.4:53                Masq    1      0          0
```

Assisted Practice



Handling Component Failure Threshold

Duration: 10 Min.

Problem statement:
You have been asked to view the nodes within a cluster and gather detailed health information for handling component failure threshold.

Outcome:
By the end of this demo, you will be able to view the nodes within a cluster and gather detailed health information to effectively handle component failure thresholds.

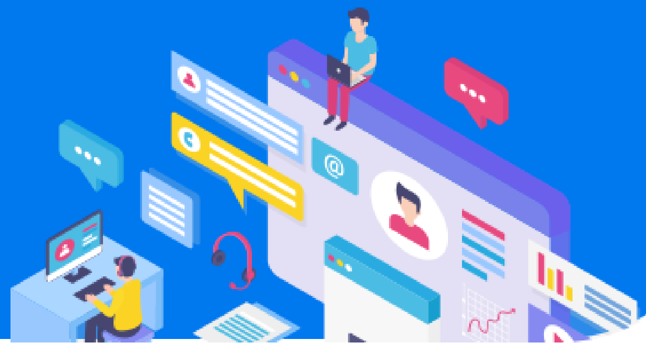
Note: Refer to the demo document for detailed steps

Assisted Practice: Guidelines

Steps to be followed:

1. Check the cluster's health information

Assisted Practice



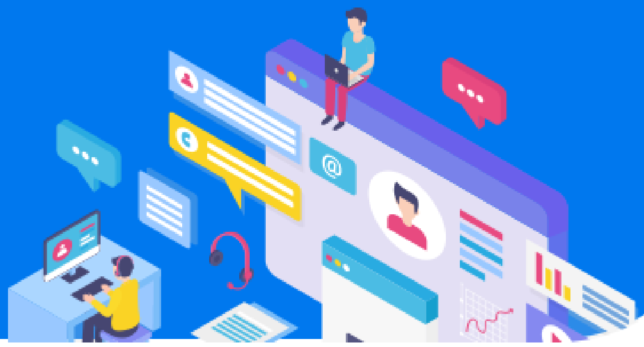
Troubleshooting Networking Issues **Duration: 10 Min.**

Problem statement:
You have been asked to create, troubleshoot, and modify an httpd-pod and its associated service in a Kubernetes environment.

Outcome:
By the end of this demo, you will be able to create, troubleshoot, and modify an HTTP pod and its associated service in a Kubernetes environment.

Note: Refer to the demo document for detailed steps

Assisted Practice: Guidelines



Steps to be followed:

1. Create an httpd-pod
2. Create an httpd-service
3. Check the labels for all the pods

Quick Check

You are troubleshooting a service issue in your Kubernetes cluster. You need to check the IP address assigned to the service within the cluster. To do this, you decide to list all services running in all namespaces and locate the Cluster IP of the service.
Which command will you run to find the Cluster IP?

A. kubectl get pod

B. kubectl get service

C. kubectl describe node

D. kubectl get deployment

Case Studies

Case Study: ING

Location: Amsterdam, Netherlands

Industry: Finance

Challenge

After undergoing an Agile transformation, ING realized it needed a standardized platform to support its developers' work.

Solution

The company's team could build an internal public cloud for its CI/CD pipeline and green-field applications. It can accomplish this by employing Kubernetes for container orchestration and Docker for containerization.

Case Study: AdForm

Location: Copenhagen, Denmark

Industry: AdTech

Challenge

Maintenance of VMs led to slowing down the technology, new software updates, and the self-healing process

Solution

Adoption of Kubernetes to use new frameworks

Case Study: Pinterest

Location: San Francisco, California, USA

Industry: Web and Mobile

Challenge

Building the fastest path from an idea to production, without making engineers worry about the underlying infrastructure

Solution

Moving services to containers

Case Study: Nokia

Location: Espoo, Finland

Industry: Telecommunications

Challenge

Deliver software to several telecom operators and add the software to their infrastructure

Solution

A shift to cloud native technologies would help teams to have infrastructure-agonistic behavior in their products.

Key Takeaways

- Elements like virtual machines, pods, containers, and nodes support troubleshooting in a Kubernetes environment.
- Common logging methods for containerized applications are writing to standard output and error stream.
- Cluster-level logging enables access to application logs even if a node dies, a pod gets evicted, or a container crashes.
- For diagnosing the problem in the application, first assess whether the problem lies in the pods, replication controller, or services.

Thank You