Container Orchestration Using Kubernetes

TECHNOLOGY

Troubleshooting and Kubernetes Case Studies

simplilearn

A Day in the Life of a DevOps Engineer

You are working as a DevOps engineer in an organization. You are responsible for designing and executing solutions to use a Kubernetes cluster, configuring hardware, peripherals, services, and managing settings and storage. You're also responsible for troubleshooting difficulties that users have reported.

You need to decide how to troubleshoot a Kubernetes cluster, debug, and examine the application.

To achieve the above tasks, you will be learning a few concepts in this lesson that will help find a solution for the given scenario.

Learning Objectives

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

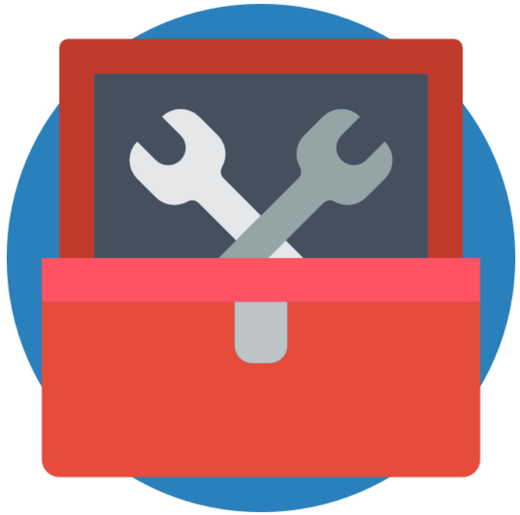
- Troubleshoot a Kubernetes cluster to ensure the optimal performance and reliability
- Configure options in Kubernetes cluster logging architecture for customizing log collection and storage
- Comprehend cluster, node-level, and container logs in more depth for optimizing Kubernetes operations
- Troubleshoot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

TECHNOLOGY

Overview of Troubleshooting in Kubernetes

Troubleshooting

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 understand issues.

The elements are:

- Kubernetes cluster
- Containerized applications

Kubernetes Cluster: Debugging a Cluster

To debug a cluster, check whether all the nodes are registered correctly. It is important to ensure that all the nodes are present in the **Ready** state.

Demo

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

kubectl get nodes
```

Kubernetes Cluster: Check the Health of the Cluster

The health of the cluster can be checked by running the following command:

Demo

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

 kubectl cluster-info

 kubectl cluster-info dump
```

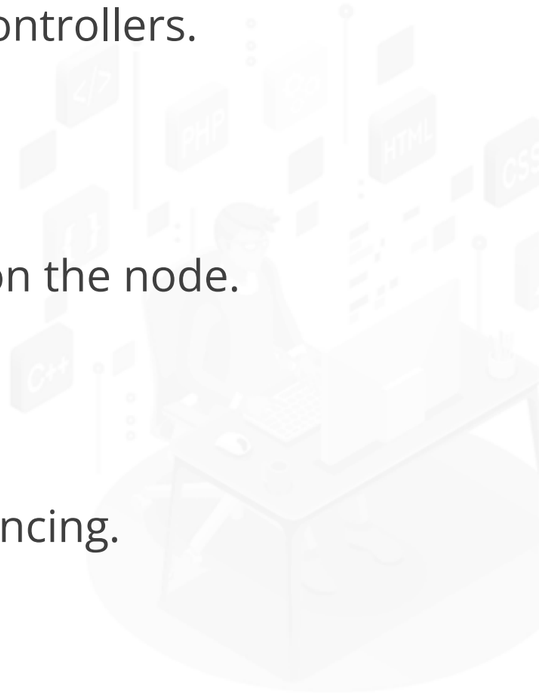
Locations of Log Files on Master Node

A thorough investigation of issues in the cluster will require analyzing the log files in the relevant machines.

/var/log/kube-apiserver.log	The API server is responsible for serving the API.
/var/log/kube-scheduler.log	The scheduler is responsible for making scheduling decisions.

Locations of Log Files on Master Node

/var/log/kube-controller-manager.log	The controller manager manages replication controllers.
/var/log/kubelet.log	Kubelet is responsible for running containers on the node.
/var/log/kube-proxy.log	Kube-proxy is responsible for service load balancing.



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 Failūre Scenarios

Following are the specific cluster failure scenarios:

1. API server crashes
2. APE backing storage lost
3. Supporting services crash
4. Individual node shutdown
5. Network partition
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
- Must use high availability configuration
- Snapshot apiserver PDs or EBS volumes periodically
- Use Replication Controller and services
- Design apps to tolerate unexpected restarts

Troubleshooting Kubernetes Cluster

Duration: 20 mins

Problem Statement:

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

ASSISTED PRACTICE

Assisted Practice: Guidelines

Steps to be followed:

1. Troubleshoot using dumps

Kubernetes Cluster Logging Architecture

Application Logs

It helps 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:

1. Write to standard output stream
2. Write to standard error stream

Cluster-Level Logging

When logs have separate storage that is independent of containers, pods, or nodes in a cluster, it is known as cluster-level logging.

Cluster-level logging architecture:

Enable access to application logs even if a node dies, a pod gets evicted, or a container crashes.

Require a separate backend to store, analyze, and query logs.

Basic Logging in Kubernetes

The example shown here uses a pod specification with a container to write text to the standard output stream once every second:

Demo

```
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

To run the pod for writing text to the standard output stream, use the following command:

Demo

```
# 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
```

Fetch Logs in Kubernetes

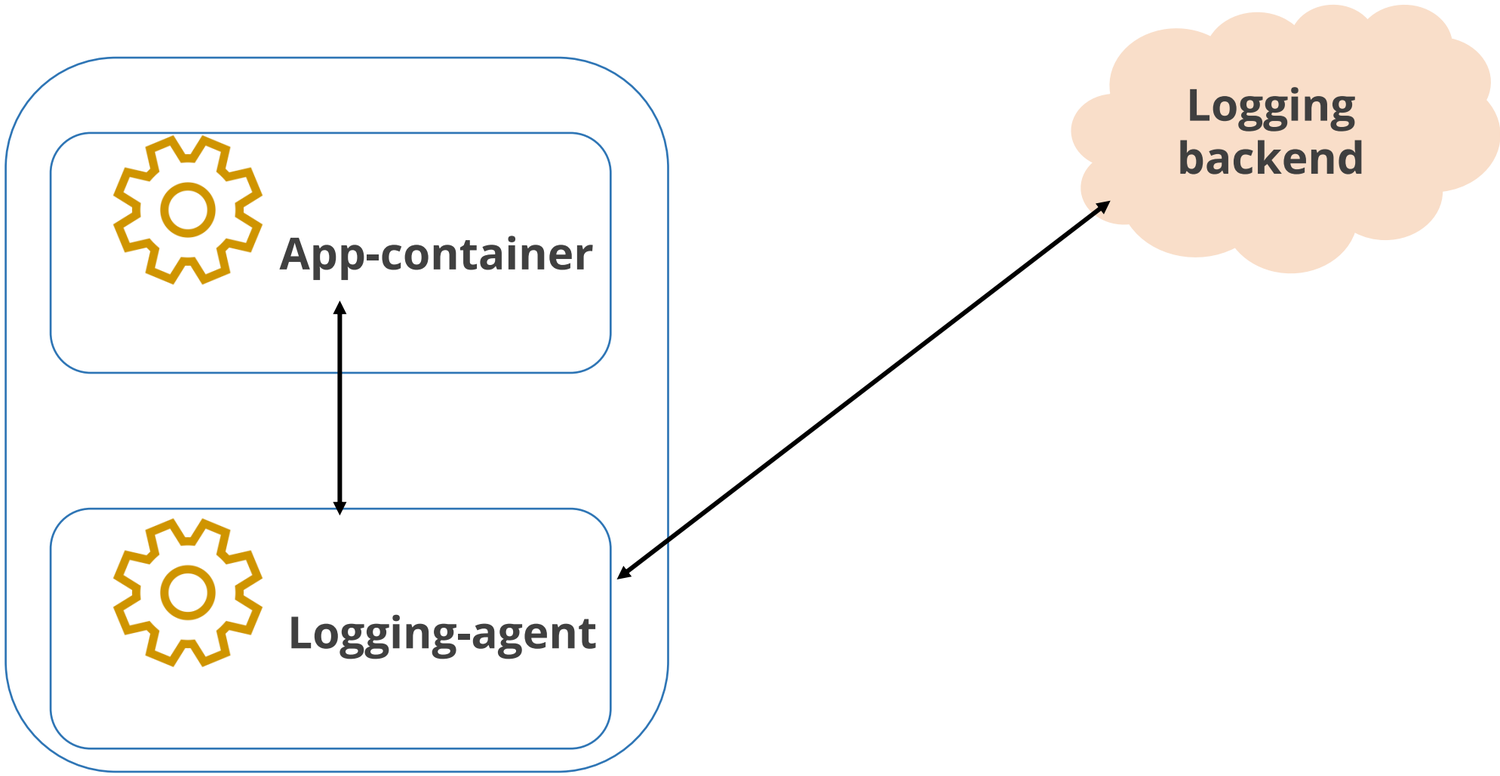
To fetch logs and retrieve them from a previous instantiation of a container, use the following commands:

Demo

```
# 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 with the application if the node-level logging agent is not flexible.



Configuration Files to Implement Sidecar Container

Here is a configuration file to implement a sidecar container with a logging agent:

Demo

```
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>
```

Demo

```
<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 second configuration file describes a pod that has a sidecar container running **fluentd**.

Demo

```
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.

Demo

```
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
```

Understanding the Kubernetes Cluster Logging Architecture

Duration: 15 mins

Problem Statement:

You have been asked to monitor, troubleshoot, and manage applications and services within the Kubernetes clusters.

ASSISTED PRACTICE

Assisted Practice: Guidelines

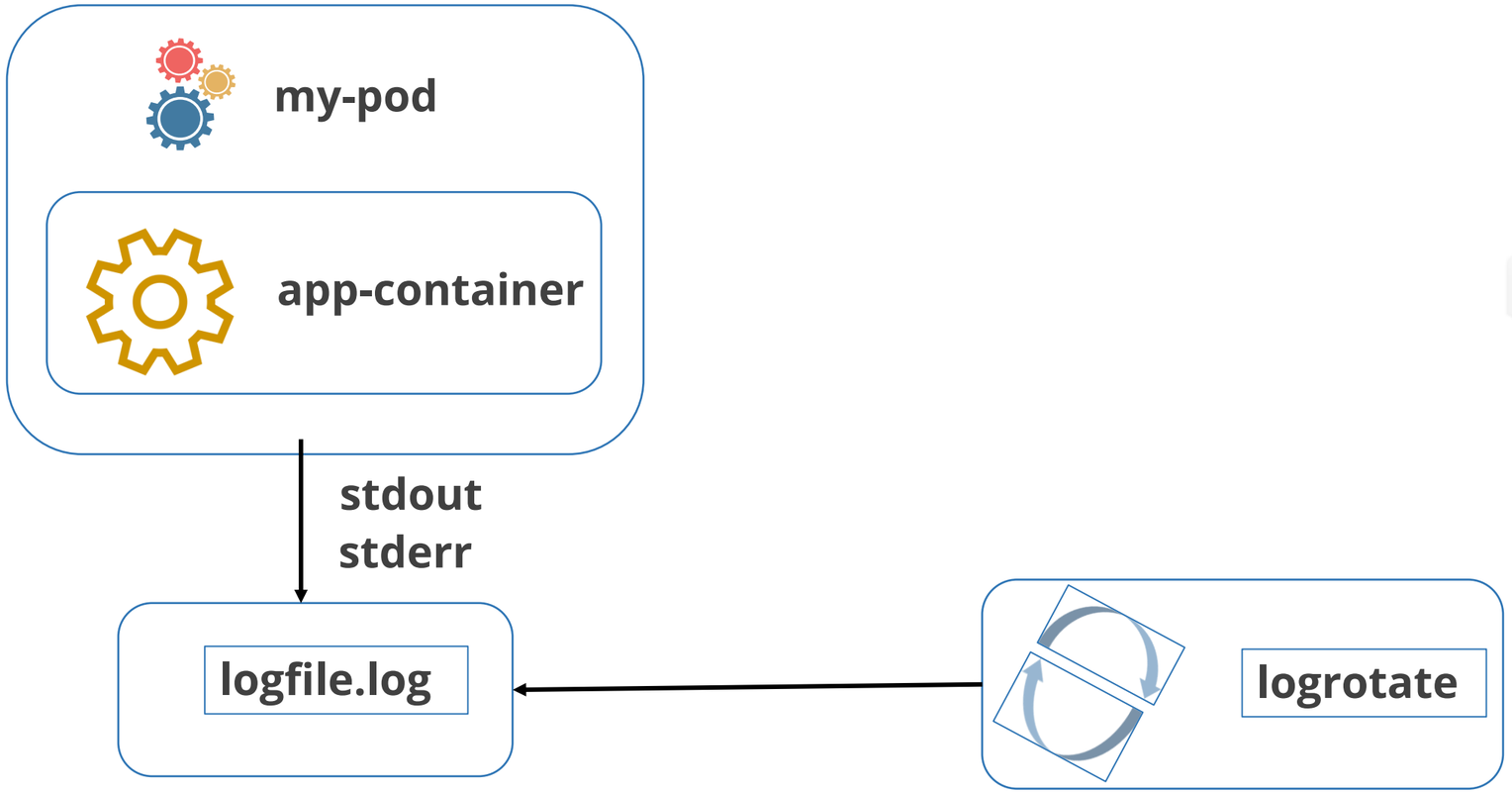
Steps to be followed:

1. Get help with logging
2. Understand log options and switch information

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 implement log rotation.



CRI Container Runtime

The kubelet is responsible for managing the logging directory structure and rotating the logs while using a CRI container runtime.

There are two kubelet flags that can be used:

- 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

System Component Logs

There are two types of system components:

Those that run a container

Example: Kubernetes Scheduler and kube-proxy

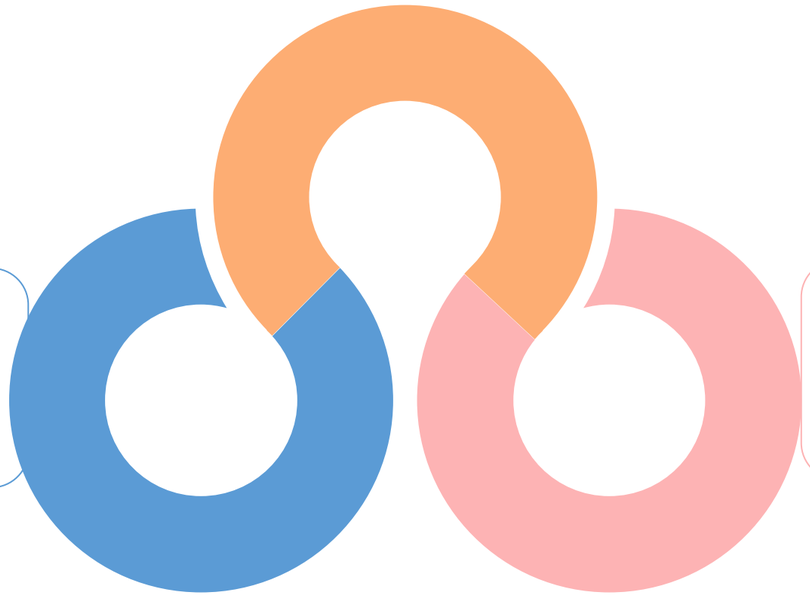
Those that do not run a container

Example: Kubelet and Container runtime

Methods of Cluster-Level Logging

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

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

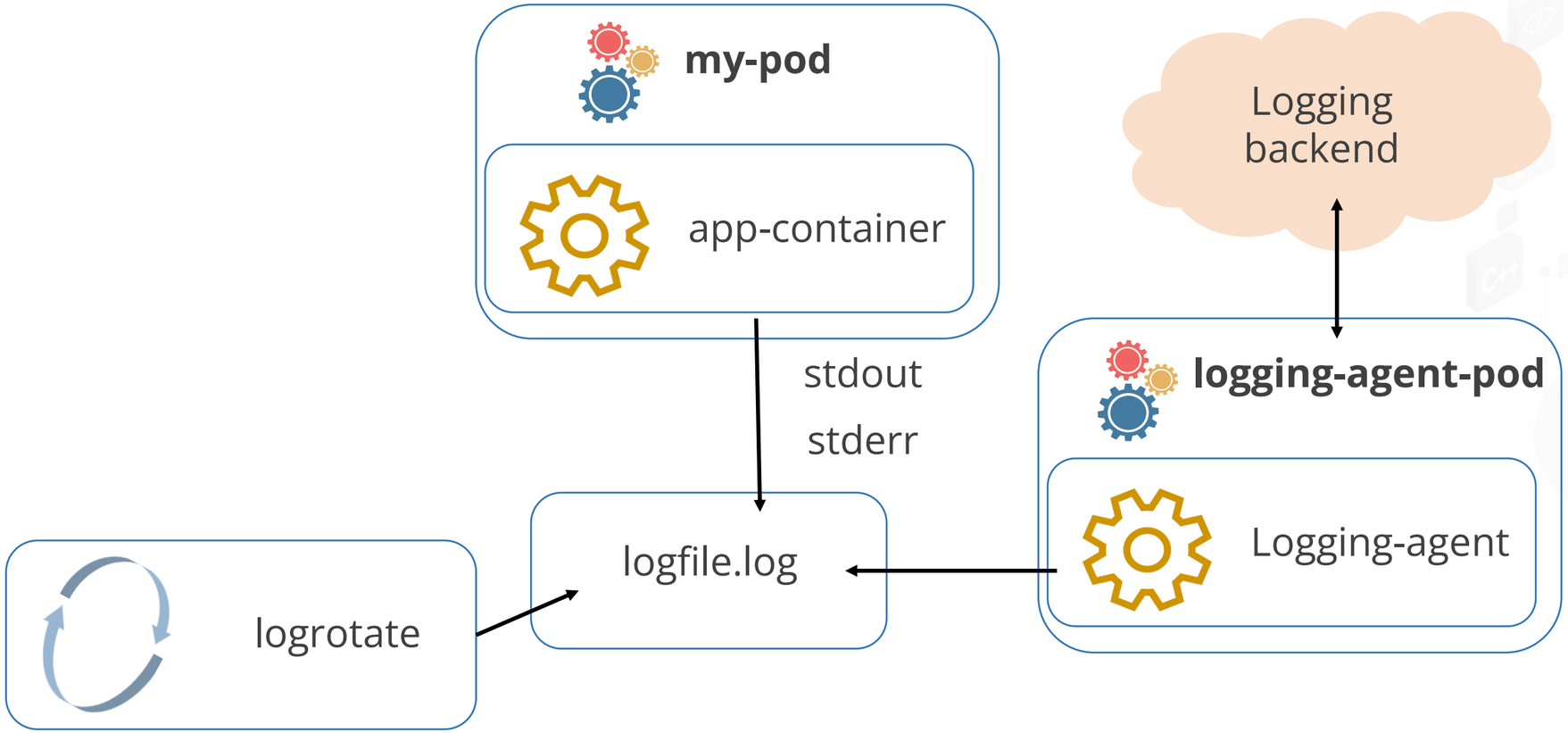


Using a dedicated sidecar container for logging in an application pod

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 on each node. A logging agent is a tool that pushes logs to a backend.



Usage of Sidecar Container

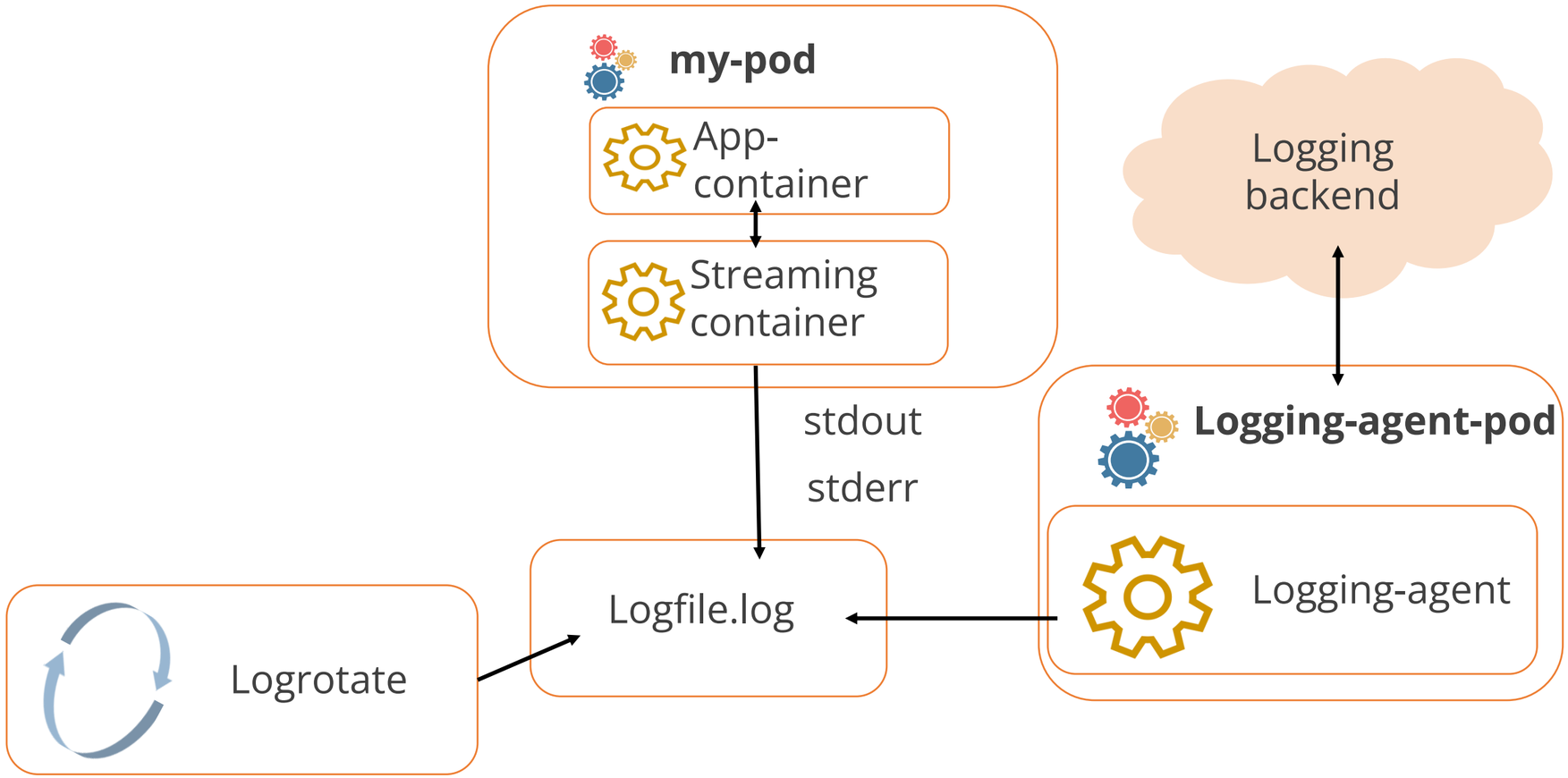
A sidecar container can be used for cluster-level logging in either of the following ways:

Stream application logs to its own stdout

Run 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 own **stdout** or **stderr** stream.



Pod with Two Sidecar Containers

Here is a configuration file for a pod with a single container:

Demo

```
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
```

Demo

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

Pod with Two Sidecar Containers

Here is a configuration file for a pod that has two sidecar containers:

Demo

```
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
```

Demo

```
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 you run the pod, each log stream can be accessed separately by running the following command:

Demo

```
# 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
```

Understanding Cluster and Node Logs

Duration: 10 mins

Problem Statement:

You have been asked to understand the procedure of inspecting and troubleshooting the control-plane components like the API server, controller manager, etcd, and kubelet service in worker nodes.

ASSISTED PRACTICE

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

Troubleshooting Node Readiness

Duration: 15 mins

Problem Statement:

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

.

ASSISTED PRACTICE

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

Container Logs

Fetch Container Logs

Problems in a cluster can happen at the container level. Kubectl provides the following command to fetch the logs.

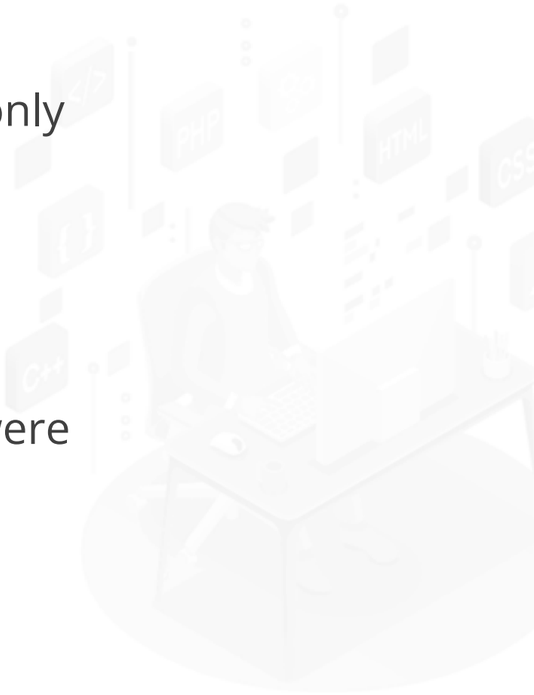
kubectl logs counter

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

The Container logs command retrieves logs present at the time of executing the command.

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

Docker Container Command Options

Name	Description
--details	Show extra details provided to logs
--follow, --f	Follow log output
--since	Display logs since timestamp 20%
--tail <number>	Number of lines to show from the end of the logs
--until	Show logs before a timestamp

Understanding Container Logs

Duration: 15 mins

Problem Statement:

You have been asked to view and check the container logs within a Kubernetes cluster using the crictl commands.

ASSISTED PRACTICE

Assisted Practice: Guidelines

Steps to be followed:

1. Check the container logs using the crictl commands

Analyzing Pod Logs

Duration: 10 mins

Problem Statement:

You have been assigned a task to create and check pod logs.

ASSISTED PRACTICE

Assisted Practice: Guidelines

Steps to be followed:

1. Configure and verify Nginx deployment

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:

Demo

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

kubectl describe pods ${POD_NAME}
```

Pods in Pending State

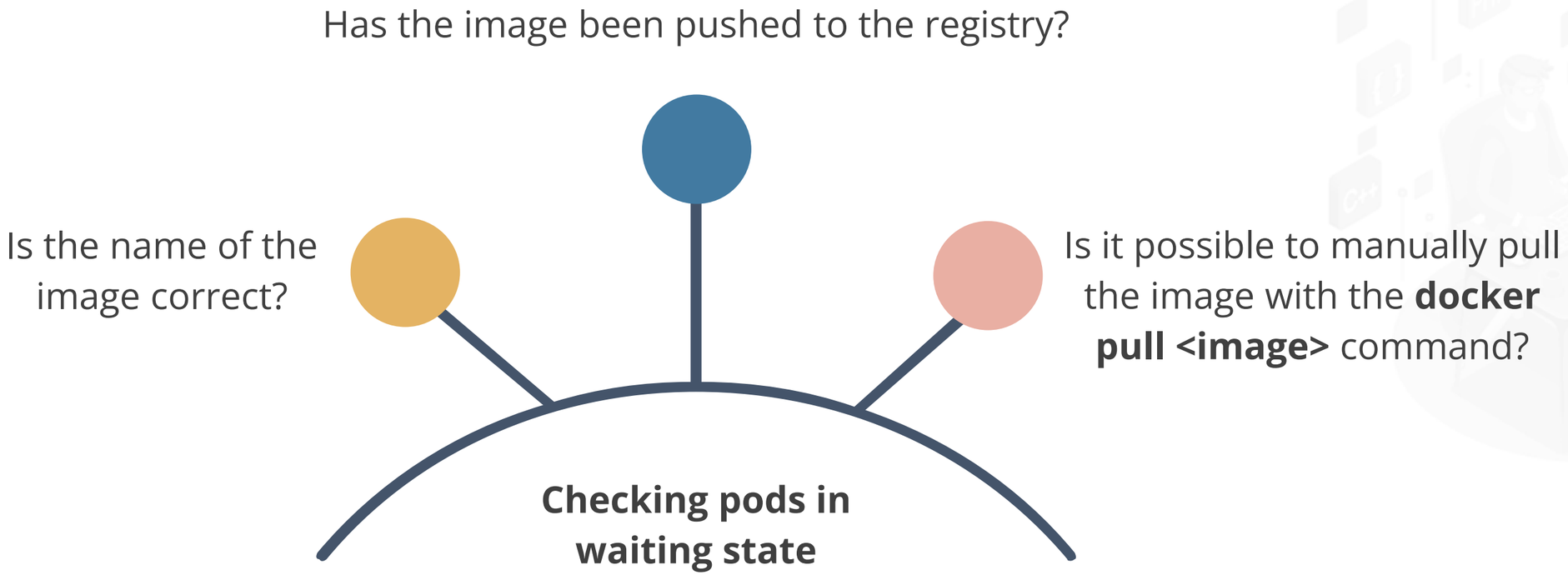
When a pod gets stuck in a pending state, it indicates its inability to be scheduled onto a node. There could be two reasons for this:

1. There are not enough resources: the supply of CPU and memory in the cluster is depleted.

2. The pod is 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.



Pods Not Behaving as Expected

If the pod is not behaving as expected, there could be two reasons:

- 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 perform the function of providing load balancing across a set of pods. Service problems could be debugged in the following ways:

- 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 endpoints should match the number of pods. To check whether endpoints are available for the service, use the following command:

Demo

```
# command to view the endpoints

kubectl get endpoints
```

Understanding Application Troubleshooting

Duration: 10 mins

Problem Statement:

You have been asked to set up an application pod in Kubernetes, diagnosing potential issues, and implementing necessary corrections to ensure its successful deployment.

ASSISTED PRACTICE

Assisted Practice: Guidelines

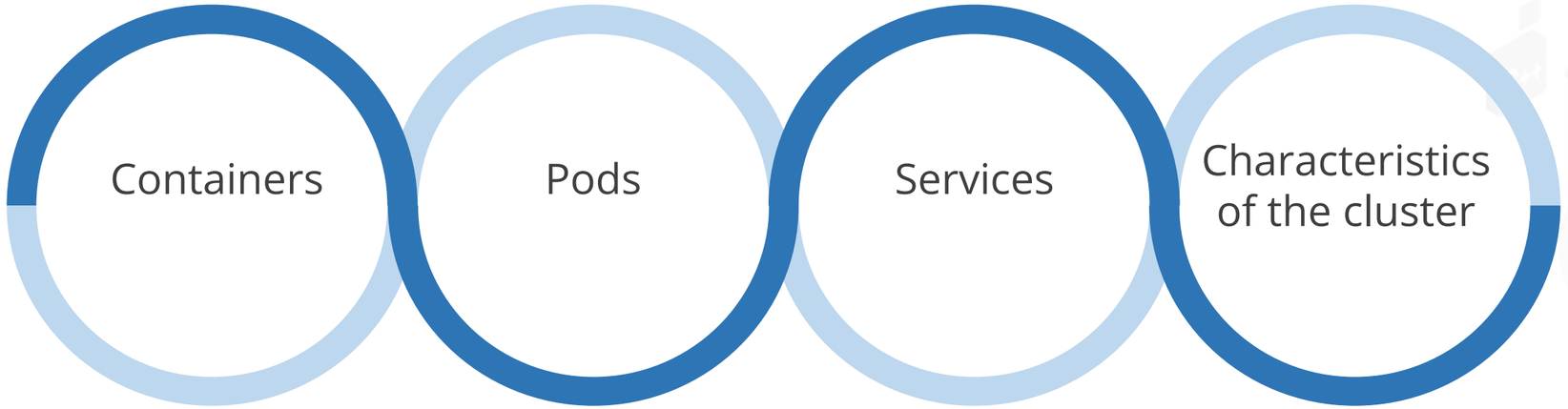
Steps to be followed:

1. Setup and diagnose the application pod

Monitoring Tools

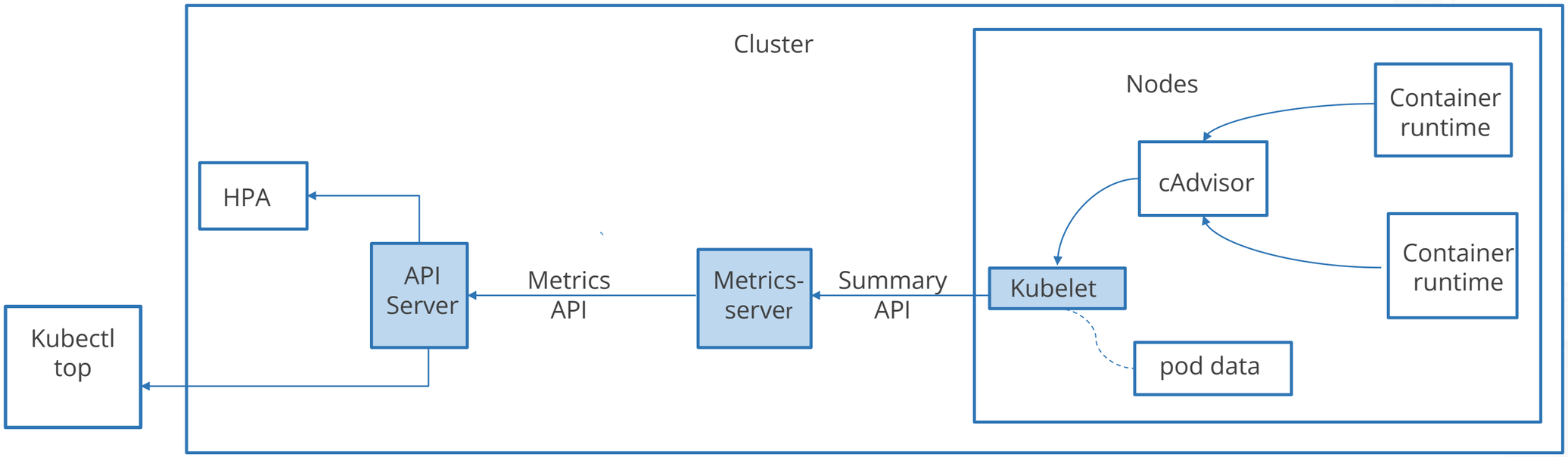
Elements for Monitoring Resources

Kubernetes gives detailed information about the resource usage of an application. Check the performance of the application in a Kubernetes cluster by examining the following:



Architecture of Resource Metrics Pipeline

The Kubernetes metrics API provides essential metrics for auto-scaling and related use cases. The diagram below depicts the resource monitoring and scaling system by Kubernetes:



Architecture of Resource Metrics Pipeline

From right to left, the figure depicts following the architectural components of Kubernetes resource metrics pipeline

cAdvisor

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

kubelet

Node agent for container resource management. The /metrics/resource and /stats kubelet API endpoints provide access to resource metrics.

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

Metrics API

The metrics API gives the number of resources currently used by a specific node or pod.

1

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.

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:

Demo

```
# 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:

Demo

```
# 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:

Demo

```
# 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

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

Demo

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

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

Output

14552
```

Inspect the IP Table Rules

To inspect the IP table rules, run the **IP table** command:

Demo

```
# 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

To list the translation table of IPs, use the following command:

Demo

```
# 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
```

Handling Component Failure Threshold

Duration: 10 mins

Problem Statement:

You have been asked to view the nodes within a cluster and gather detailed health information for ensuring a proper functioning of the nodes.

ASSISTED PRACTICE

Assisted Practice: Guidelines

Steps to be followed:

1. Check the cluster health information

Troubleshooting Networking Issues

Duration: 15 mins

Problem Statement:

You have been asked to understand the process of creating, troubleshooting, and modifying an httpd-pod and its associated service in a Kubernetes environment.

ASSISTED PRACTICE

Assisted Practice: Guidelines

Steps to be followed:

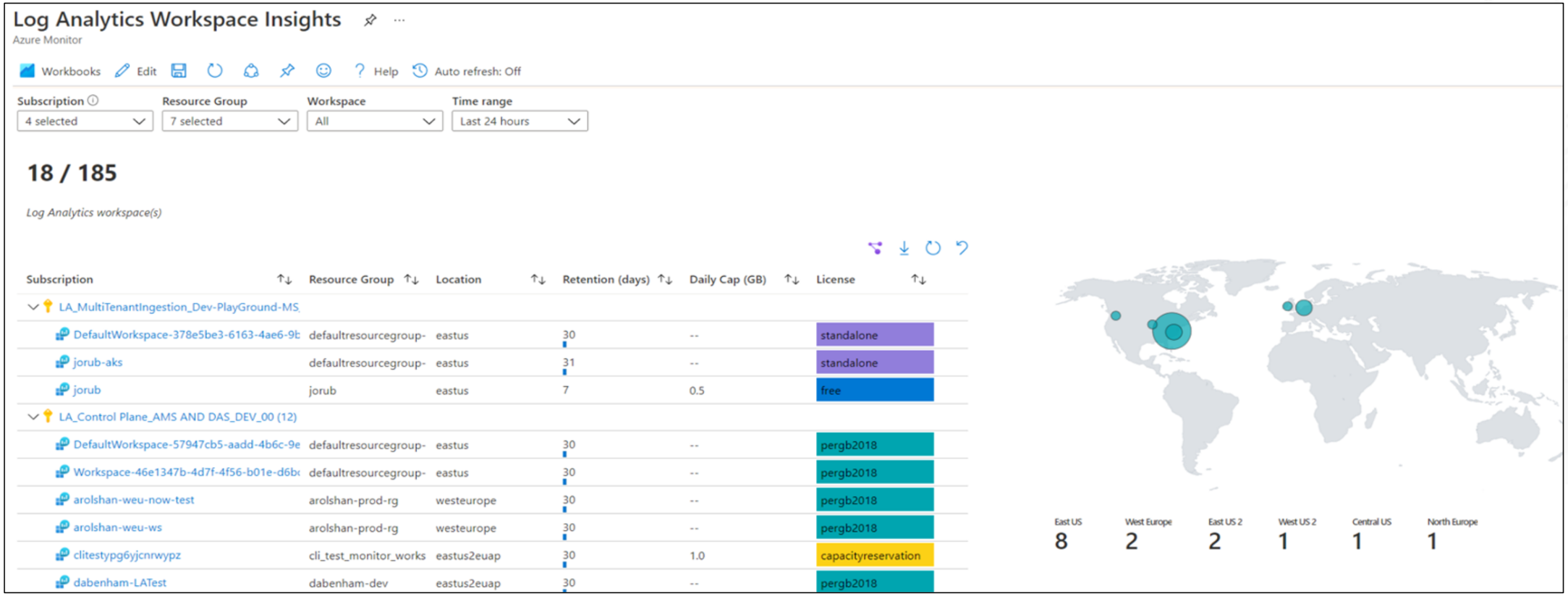
1. Create an httpd pod
2. Create an httpd service
3. Check labels for all the pods

TECHNOLOGY

AKS Monitoring and Logging

Log Analytics Workspace

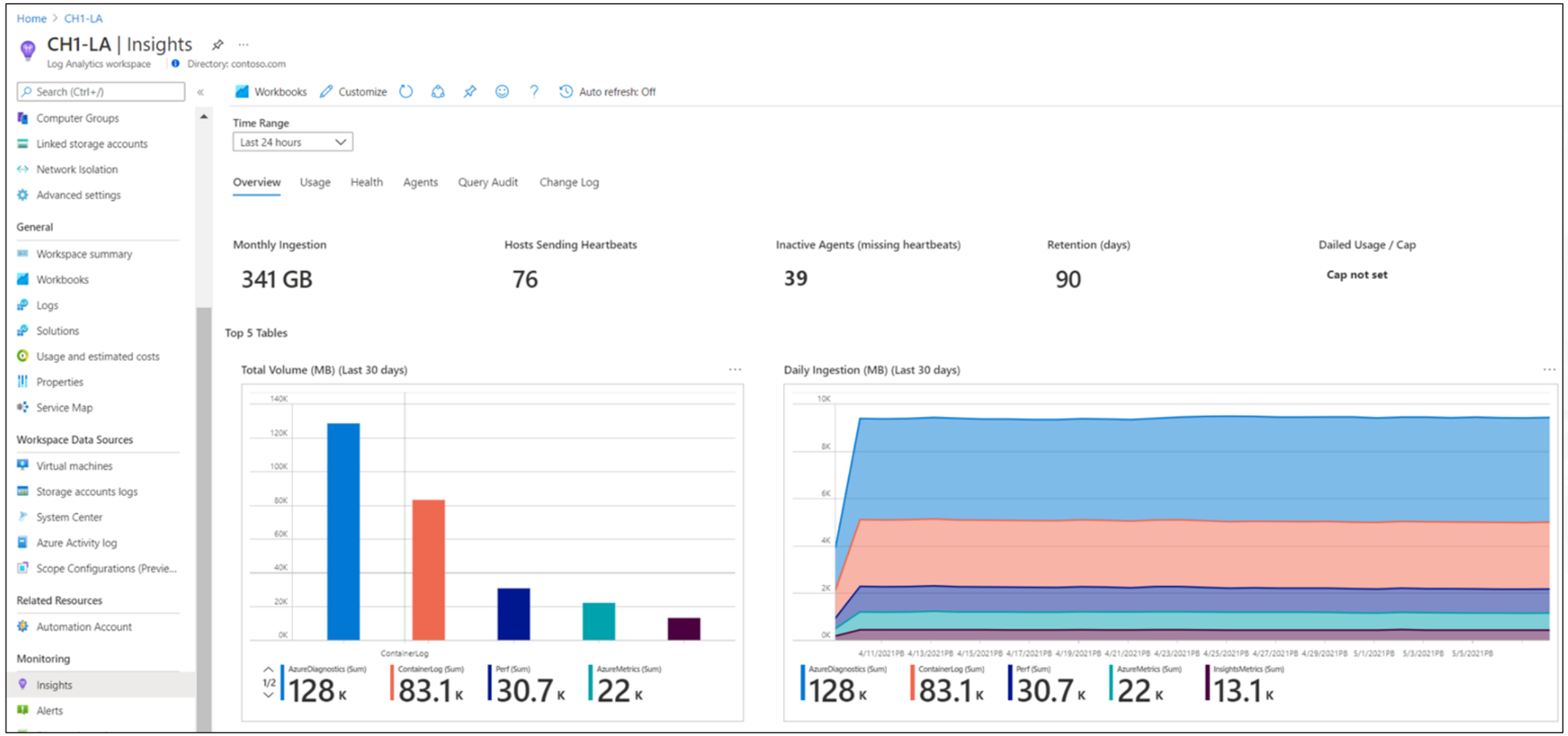
Log Analytics Workspace Insights (preview) provides comprehensive monitoring of the workspaces. It provides a unified view of the workspace usage, performance, health, agent, queries, and change log.



https://docs.microsoft.com/en-us/azure/azure-monitor/logs/media/log-analytics-workspace-insights-overview/at-scale.png

Log Analytics Workspace

It helps manage multiple clusters/nodes and monitors the performance of each of these components.



https://docs.microsoft.com/en-us/azure/azure-monitor/logs/media/log-analytics-workspace-insights-overview/at-resource.png

Azure Monitor

- Stores log data in a Log Analytics workspace
- Stores the data of all the functioning components of the AKS Cluster
- Provides an RBAC to limit access to sensitive data and maintain security

TECHNOLOGY

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.

TECHNOLOGY

Thank You

simplilearn