Certified Kubernetes Administrator

simplilearn

TECHNOLOGY

Troubleshooting and Kubernetes Case Studies

simplilearn

Learning Objectives

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

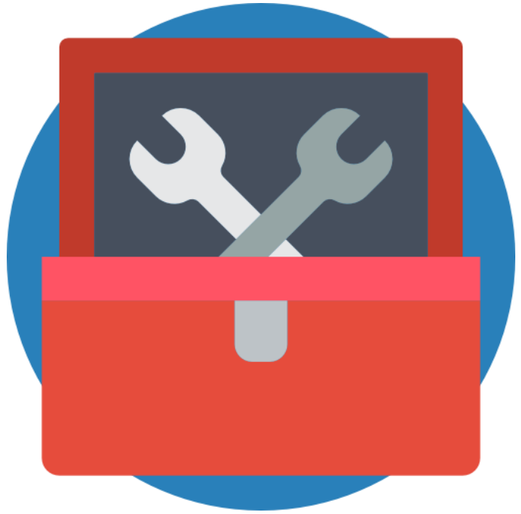
- Present an overview of Troubleshooting a Kubernetes Cluster
- List options in Kubernetes Cluster Logging Architecture
- Discuss Cluster, Node-level, and Container logs
- Debug applications
- Examine application performance

TECHNOLOGY

Overview of Troubleshooting in Kubernetes

Troubleshooting

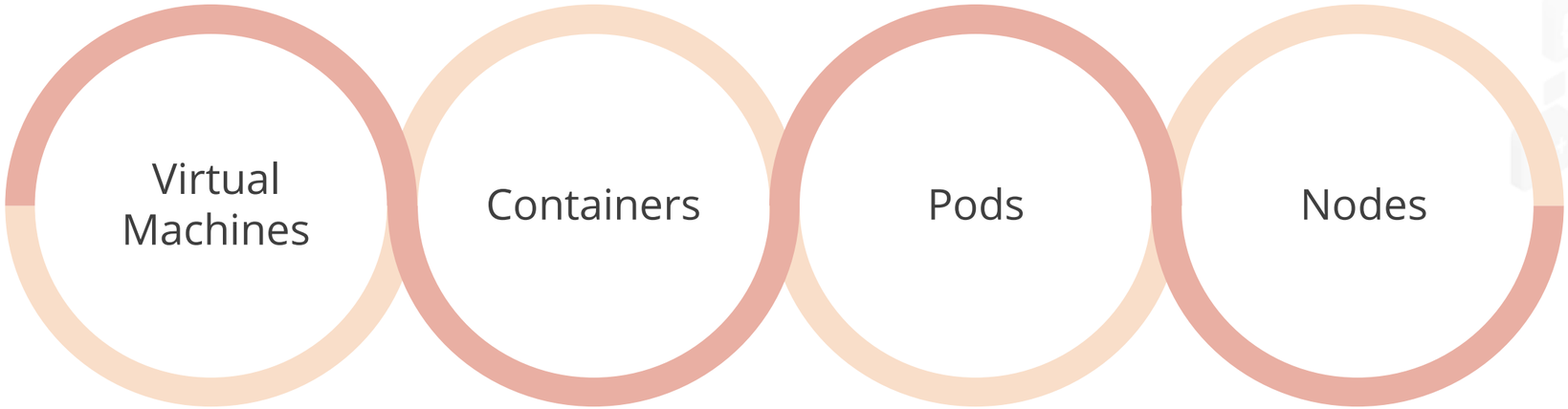
Troubleshooting in Kubernetes involves finding the root cause of a failure and taking specific steps to recover from the failure. There could be issues in any layer or component of Kubernetes.



Elements for Troubleshooting

In the Kubernetes environment, there are different elements, each of which supports debugging and troubleshooting for the administrators to analyze and understand issues.

The elements are:



List a Cluster

To debug a Cluster, the first thing to do is to check whether all the Nodes are registered correctly. Ensure that all the Nodes are present in the “Ready” state.

Demo

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

kubectl get nodes
```

Check Health of the Cluster

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

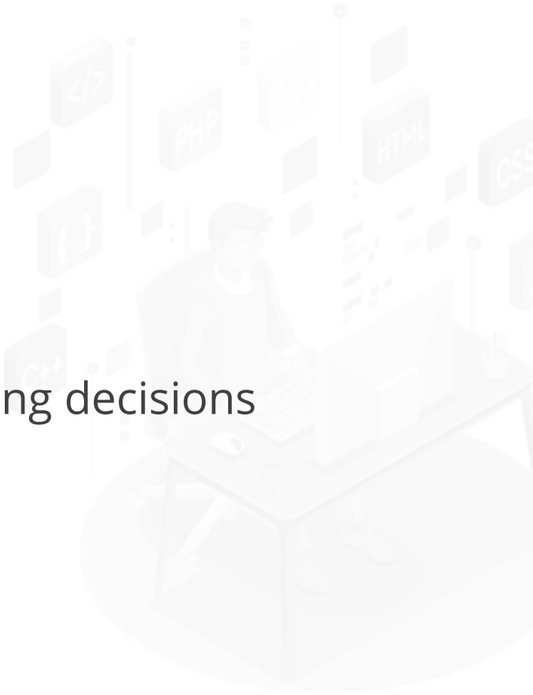
Demo

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

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	API server is responsible for serving the API
/var/log/kube-scheduler.log	Scheduler is responsible for making scheduling decisions
/var/log/kube-controller-manager.log	Controller manages Replication Controllers



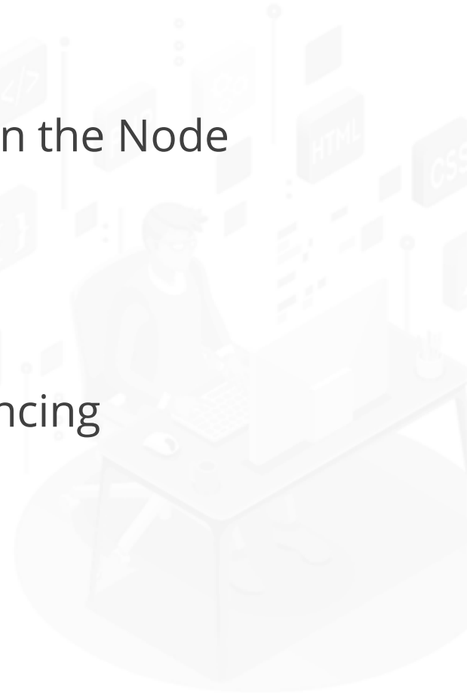
Locations of Log Files on Worker Nodes

/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

- 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

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

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

Problem Statement: Troubleshoot K8S Cluster in Kubernetes.

ASSISTED PRACTICE

Assisted Practice: Guidelines

Steps to demonstrate Troubleshooting K8S Cluster in Kubernetes:

1. Getting to know the Cluster info
2. Troubleshooting via dumps
3. Getting help on dumps
4. Getting Cluster-info dump on the specific namespace
5. Getting errors and warnings

Kubernetes Cluster Logging Architecture

Application Logs

Application logs help understand the inside of an application.

Most modern applications have a logging mechanism that help 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 a separate storage that is independent of Containers, Pods, or Nodes in a Cluster, it is known as Cluster-level logging.

Cluster-level logging architectures:

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:

```
Done

     apiVersion: v1
   Kind: pod
   Metadata:
      name: counter
   spec:
      containers:
    - name:    count
      image:   busybox
      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

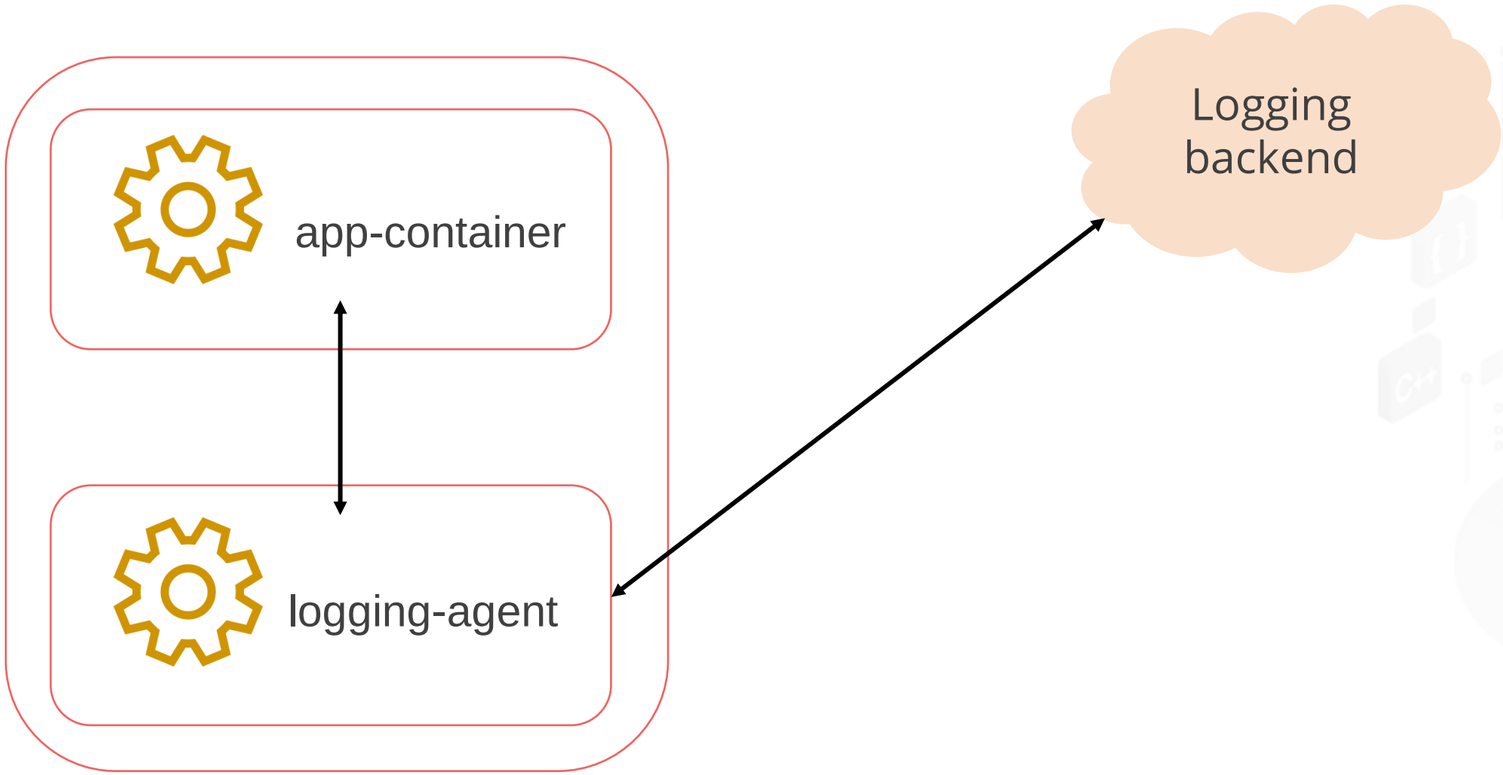
To fetch logs and retrieve them from a previous instantiation of a Container, use the following commands:

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
```

Side Container with a Logging Agent

A side 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. Fluentd can be replaced with any logging agent.

Demo

```
apiVersion: v1
kind: Pod
metadata:
  name: counter
spec:
  containers:
  - name: count
    image: busybox
    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-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 Kubernetes Cluster Logging Architecture

Problem Statement: Understand the working of Cluster Logging Architecture in Kubernetes.

ASSISTED PRACTICE

Assisted Practice: Guidelines

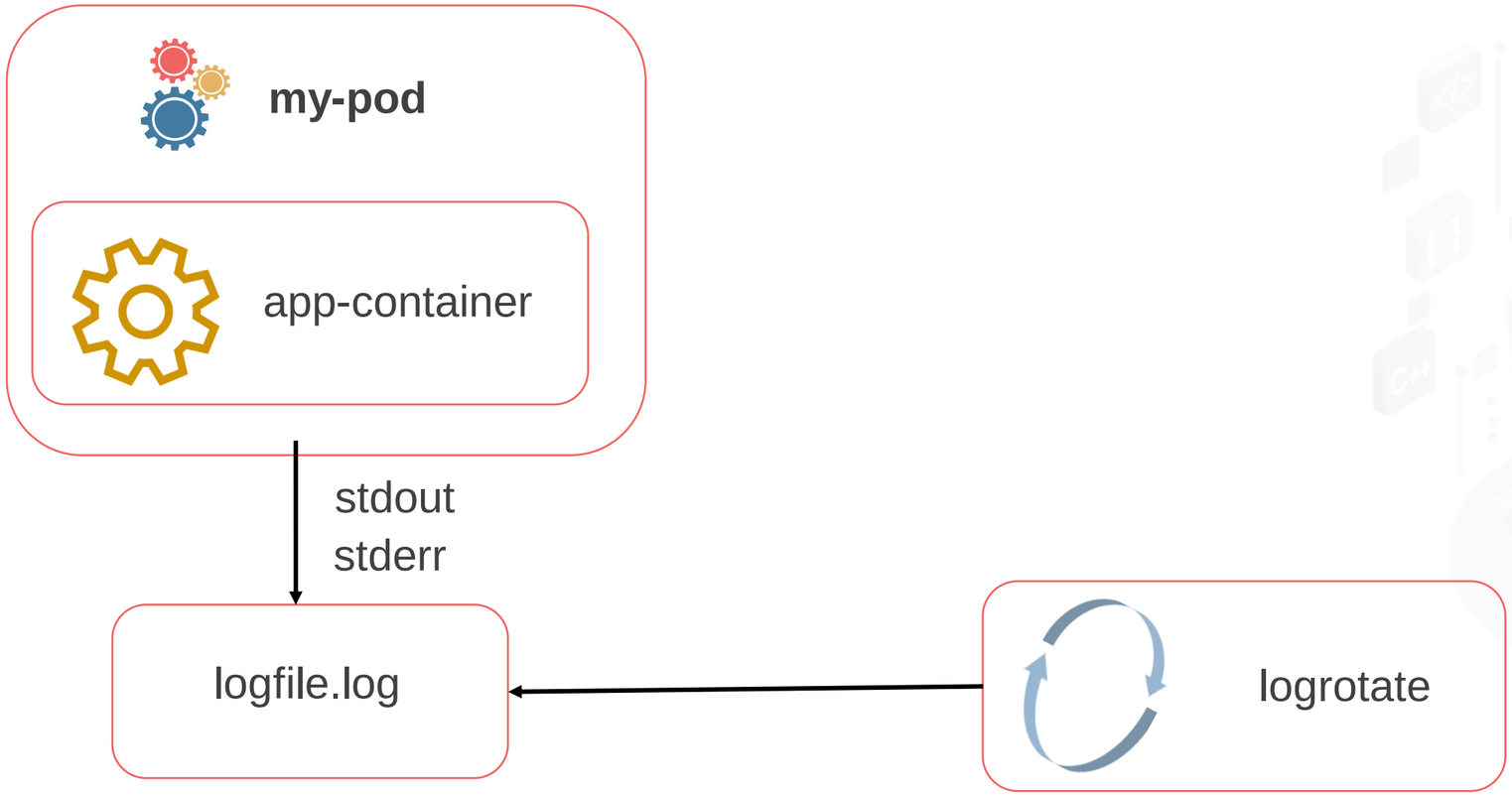
Steps to demonstrate Cluster Logging Architecture in Kubernetes:

1. Getting help on Logging
2. Conducting experiments with BusyBox
3. Logging Pods
4. Logging Deployments
5. Logging Options and Switches

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 setup 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

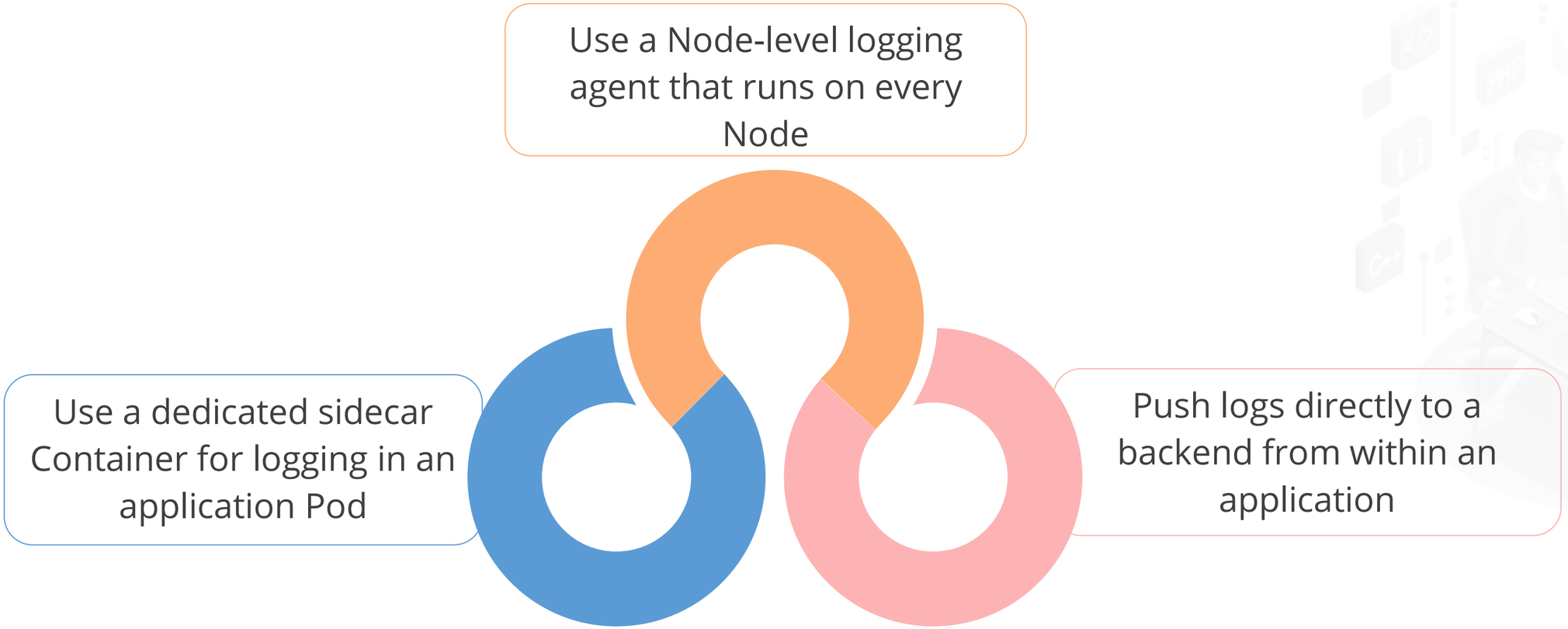
Example: Kubernetes Scheduler and kube-proxy

Those that do not run a Container

Example: Kubelet and Container runtime

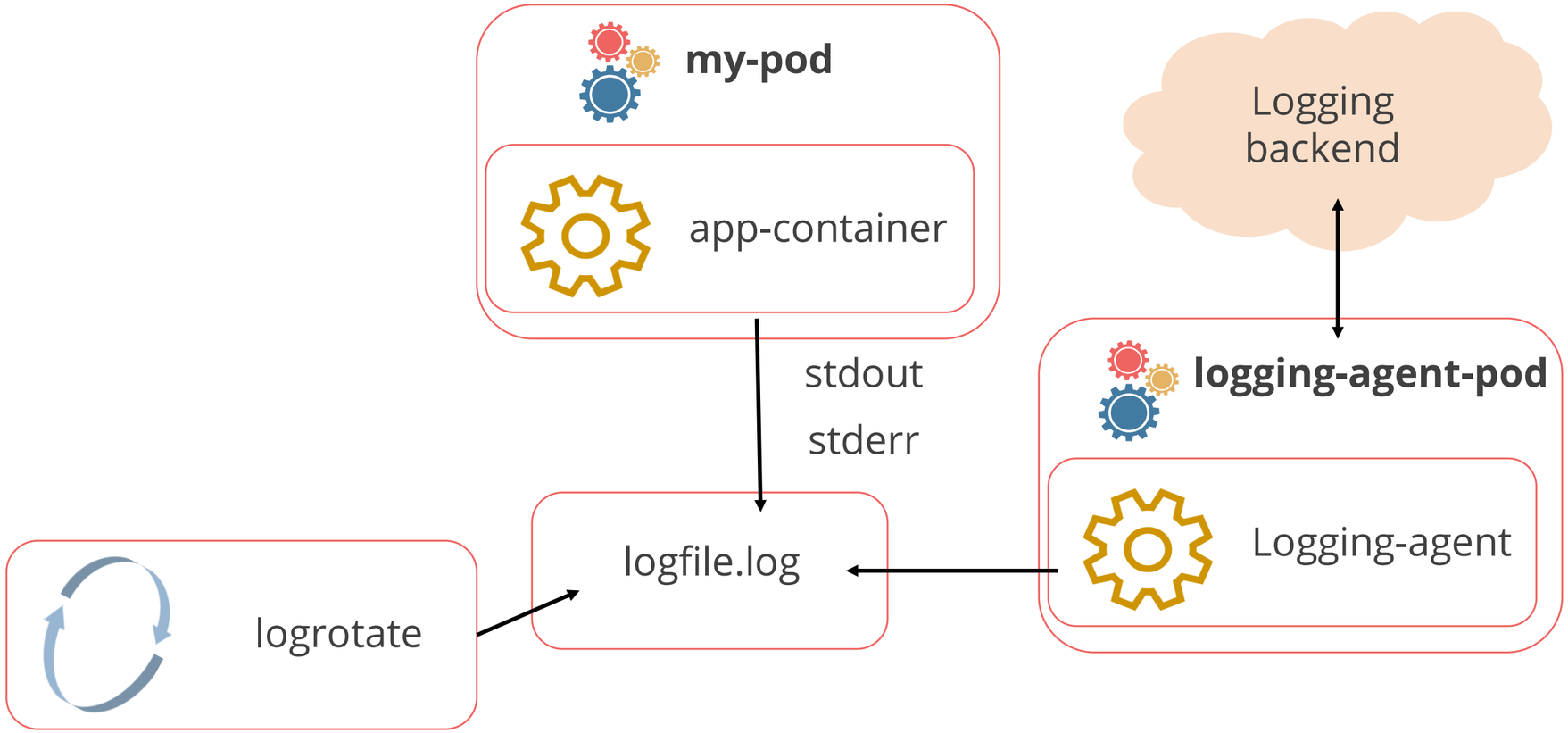
Methods of Cluster-Level Logging

Let us examine some methods that can be considered for Cluster-level logging:



Usage of Node Logging Agent

Cluster-level logging may be implemented by including a Node-level logging agent on each Node. The logging agent is a tool that pushes logs to a backend.



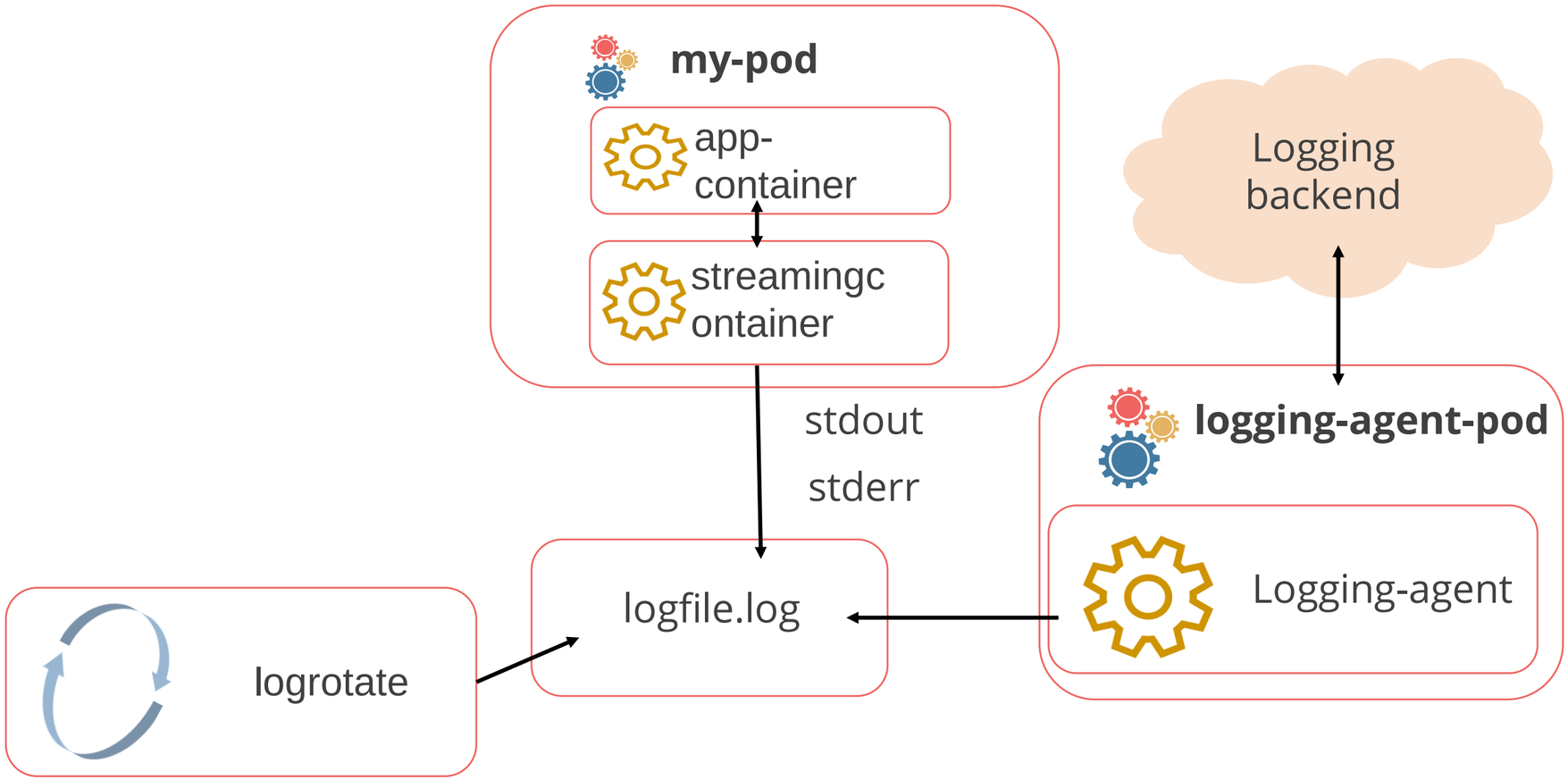
Usage of Sidecar Container

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 a Single Container

Here’s a configuration file for a Pod with a single Container:

Demo

```
apiVersion: v1
kind: Pod
metadata:
  name: counter
spec:
  containers:
  - name: count
    image: busybox
    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-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
```

Pod with Two Sidecar Containers

Here's 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
    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
```

Pod with Two Sidecar Containers

Demo

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

Problem Statement: Understand the working of Cluster and Node logs in Kubernetes.

ASSISTED PRACTICE

Assisted Practice: Guidelines

Steps to demonstrate Cluster and Node Logs in Kubernetes:

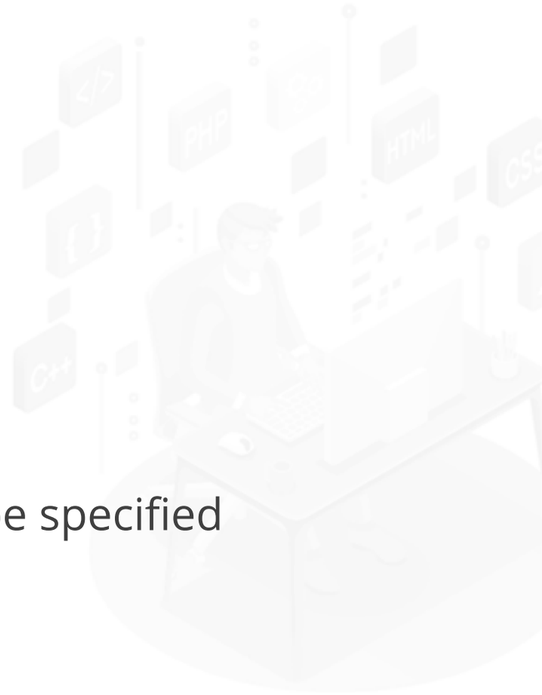
1. Create Sidecar Container

Container Logs

Fetch Container Logs

Problems in a Cluster can happen at the Container level. Kubectl provides the following command to print the logs for a Container in a Pod:

```
kubectl logs [-f] [-p] POD [-c CONTAINER]
```

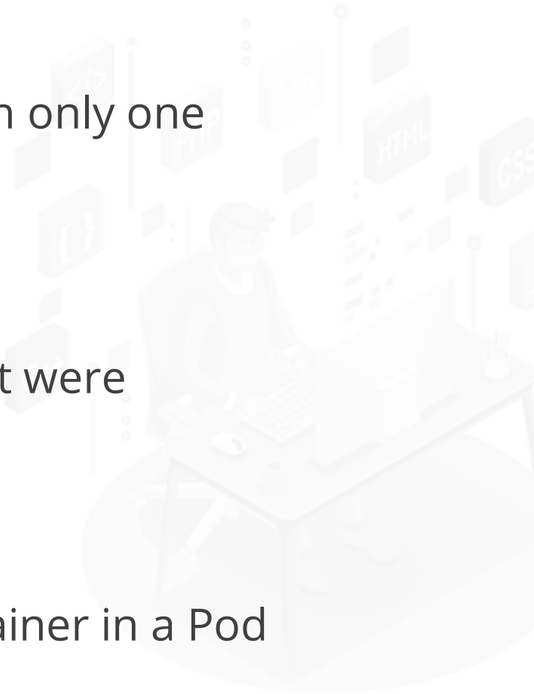


If there are many Containers in a single Pod, the names of the Containers should be specified in the command.

Commands to Fetch Container Logs

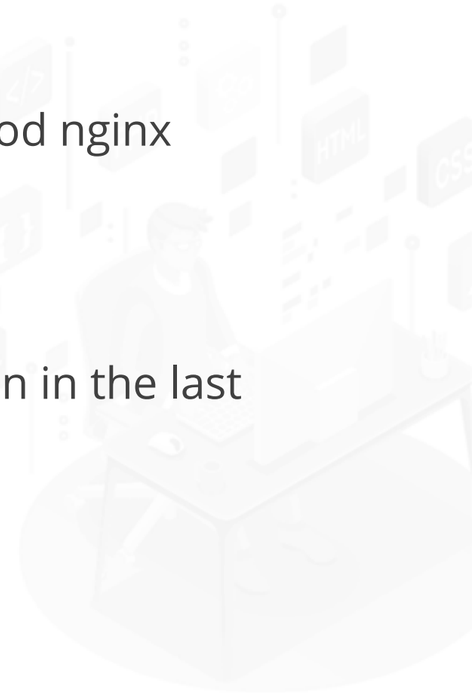
The logs of a Container can be fetched using any of the following options:

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
kubectl logs –f –c ruby web-1	To start streaming the logs of the ruby Container in a Pod called web-1



Commands to Fetch Container Logs

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 Docker Container Logs

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

```
$ docker 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 –n	Number of lines to show from the end of the logs
--until	Show logs before a timestamp

Understanding Container Logs

Problem Statement: Understand the working of Container Logs in Kubernetes.

ASSISTED PRACTICE

Assisted Practice: Guidelines

Steps to demonstrate Container Logs in Kubernetes:

1. Create Docker Container Logs
2. Create Kubernetes Container Logs

TECHNOLOGY

Application Troubleshooting

Diagnose 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
- Replication Controller

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

If a Pod gets stuck in Pending state, it means that it cannot be scheduled into a Node. There could be two reasons for this:

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

2. If the Pod is bound to a hostport—in this case, there are only a few places where the Pod can be scheduled.

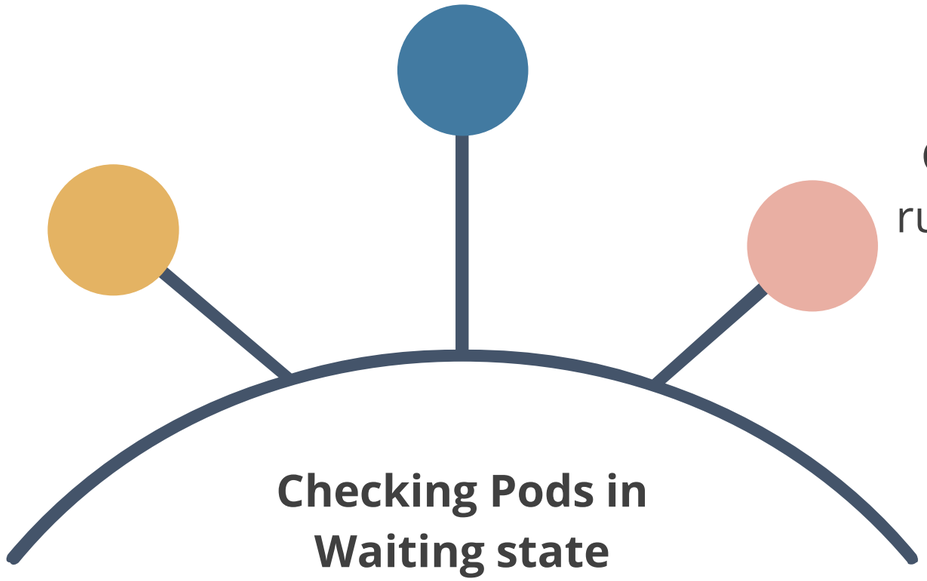
Pods in Waiting State

Failure to pull the image is one of the most common causes of Waiting Pods. In such a scenario, there are three things that need to be checked:

Has the image been pushed to the Repository?

Is the name of the image correct?

Can the image be pulled by running a manual Docker pull <image> command?



Checking Pods in Waiting state

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 Pod creation.
- A section of the Pod description is nested incorrectly or a keyname is typed incorrectly.

Debug 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 ${SERVICE_NAME}
```

Understanding Application Troubleshooting

Problem Statement: Try troubleshooting applications in Kubernetes.

ASSISTED PRACTICE

Assisted Practice: Guidelines

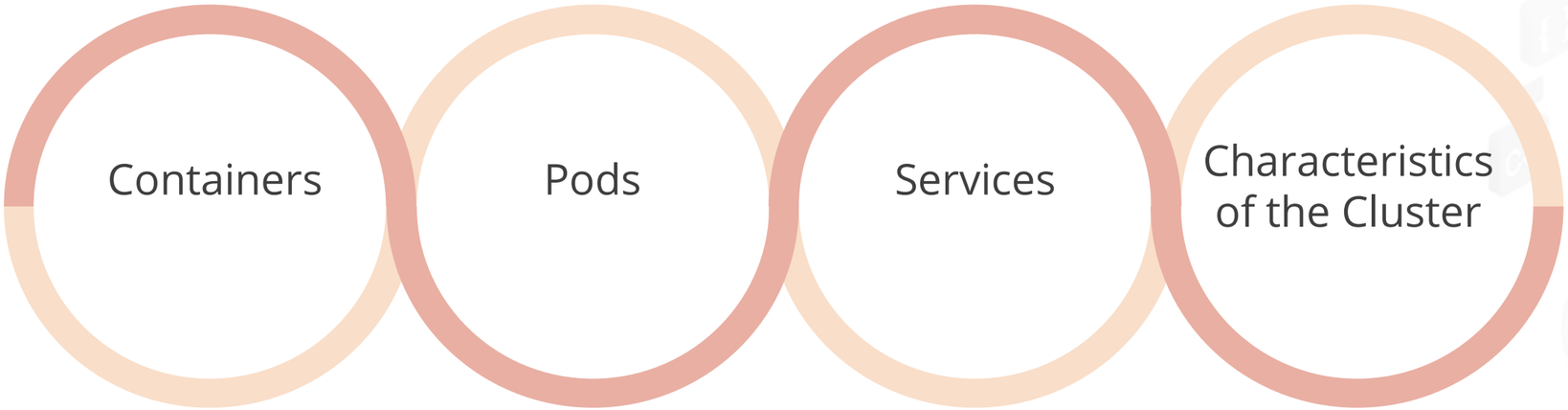
Steps to demonstrate Application Troubleshooting in Kubernetes:

1. Find pods that are not in the running state
2. Describe a Pod that is not in the running state
3. Edit the nginx version in the pod configuration
4. Apply the changes
5. Type ***sudo kubectl get pods -n test***

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:



Resource Metrics Pipeline

The Resource Metrics pipeline provides a set of metrics related to Cluster components and collected by the metrics server.

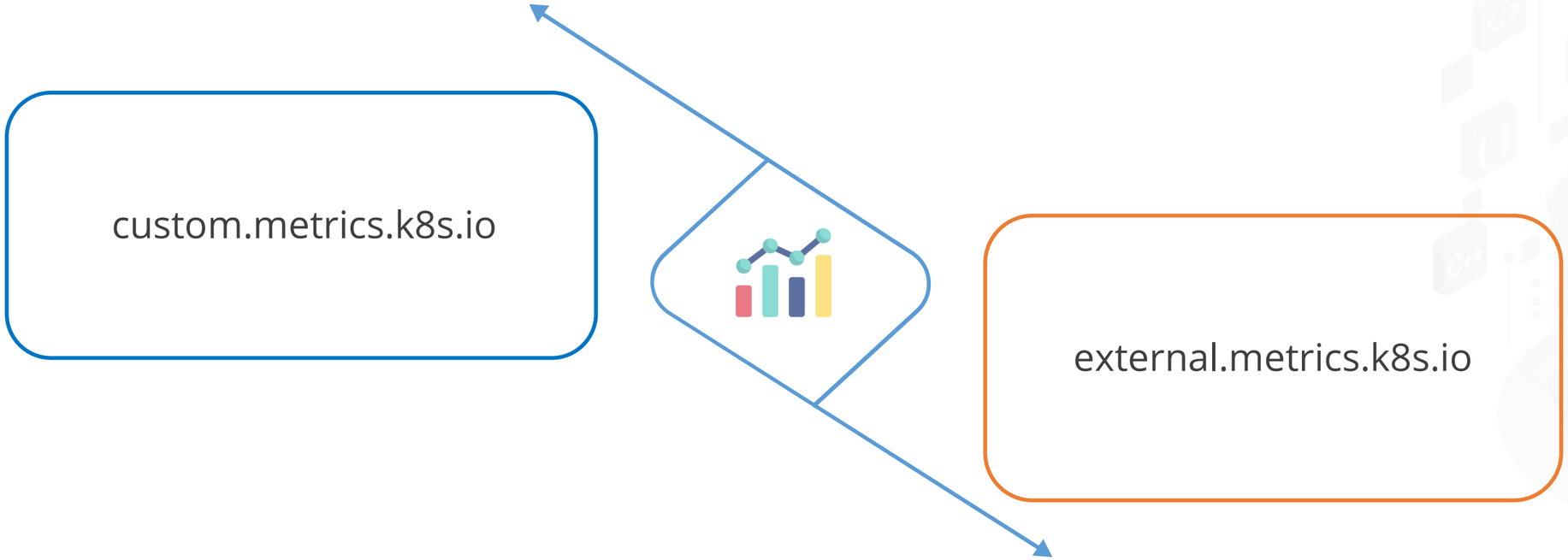
The Metrics server discovers the Nodes in the Cluster and queries each Node's kubelet to get the memory and CPU usage.

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:



Metrics API

The Metrics API gives the number of resources currently used by a specific Node or Pod.

1

The Metrics API is discoverable through the same Endpoint as other Kubernetes APIs under the path /apis/metrics.k8s.io/

2

It offers security, reliability, and scalability.

Measure Resource Usage

CPU

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

Memory

Working set is the memory that is in use and cannot be freed. Memory, measured in bytes, is the working set at the instant the metric was collected.

Metrics Server

Metrics server is the Cluster-wide aggregator of resource usage data. It collects metrics from the Summary API.

Monitoring Metrics API

Problem Statement: Install Metrics API to get familiar with monitoring tools in Kubernetes.

ASSISTED PRACTICE

Assisted Practice: Guidelines

Steps to demonstrate application troubleshooting in Kubernetes:

1. Install Metrics API
2. Type ***sudo kubectl apply -f https://github.com/kubernetes-sigs/metrics-server/releases/latest/download/components.yaml***

Commands to Debug Networking Issues

Find 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
```

Find 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 --all-namespaces


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

For Docker, first list all the Containers running on a Node.

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, 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

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

Output

14552
```

Find and Enter Pod Network Namespaces

Next, the **nsenter** command can be used to run a command in the process's network namespace.

Demo

```
# to run a command in that process's network namespace

nsenter -t your-container-pid -n ip addr
```

Find a Pod's Virtual Ethernet Interface

First, run the **nsenter** command to run a command in the process's network namespace.

Demo

```
nsenter -t your-container-pid -n ip addr
```

```
Output
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN
group default qlen 1
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
10: eth0@if11: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1450 qdisc
noqueue state UP group default
    link/ether 02:42:0a:f4:03:04 brd ff:ff:ff:ff:ff:ff link-netnsid
0
    inet 10.244.3.4/24 brd 10.244.3.255 scope global eth0
       valid_lft forever preferred_lft forever
```

Find a Pod's Virtual Ethernet Interface

Next, run the **ip addr** command in the Pod's network namespace.

Demo

```
# command to run in node's default namespace

ip addr

Output
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN
group default qlen 1
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever

. . .
```

Find a Pod's Virtual Ethernet Interface

Demo

```
7: veth77f2275@if6: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1450 qdisc
noqueue master docker0 state UP group default
    link/ether 26:05:99:58:0d:b9 brd ff:ff:ff:ff:ff:ff link-netnsid 0
    inet6 fe80::2405:99ff:fe58:db9/64 scope link
       valid_lft forever preferred_lft forever
9: vethd36cef3@if8: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1450 qdisc
noqueue master docker0 state UP group default
    link/ether ae:05:21:a2:9a:2b brd ff:ff:ff:ff:ff:ff link-netnsid 1
    inet6 fe80::ac05:21ff:fea2:9a2b/64 scope link
       valid_lft forever preferred_lft forever
11: veth4f7342d@if10: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1450
qdisc noqueue master docker0 state UP group default
    link/ether e6:4d:7b:6f:56:4c brd ff:ff:ff:ff:ff:ff link-netnsid 2
    inet6 fe80::e44d:7bff:fe6f:564c/64 scope link
       valid_lft forever preferred_lft forever
```

Inspect IP Table Rules

To inspect 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

Problem Statement: Learn how to handle component failure threshold in Kubernetes.

ASSISTED PRACTICE

Assisted Practice: Guidelines

Steps to demonstrate component threshold failure in Kubernetes:

1. List Cluster
2. Check Nodes
3. Check Cluster health

Troubleshooting Networking Issues

Problem Statement: Troubleshoot network issues in Kubernetes.

ASSISTED PRACTICE

Assisted Practice: Guidelines

Steps to demonstrate networking issues in Kubernetes:

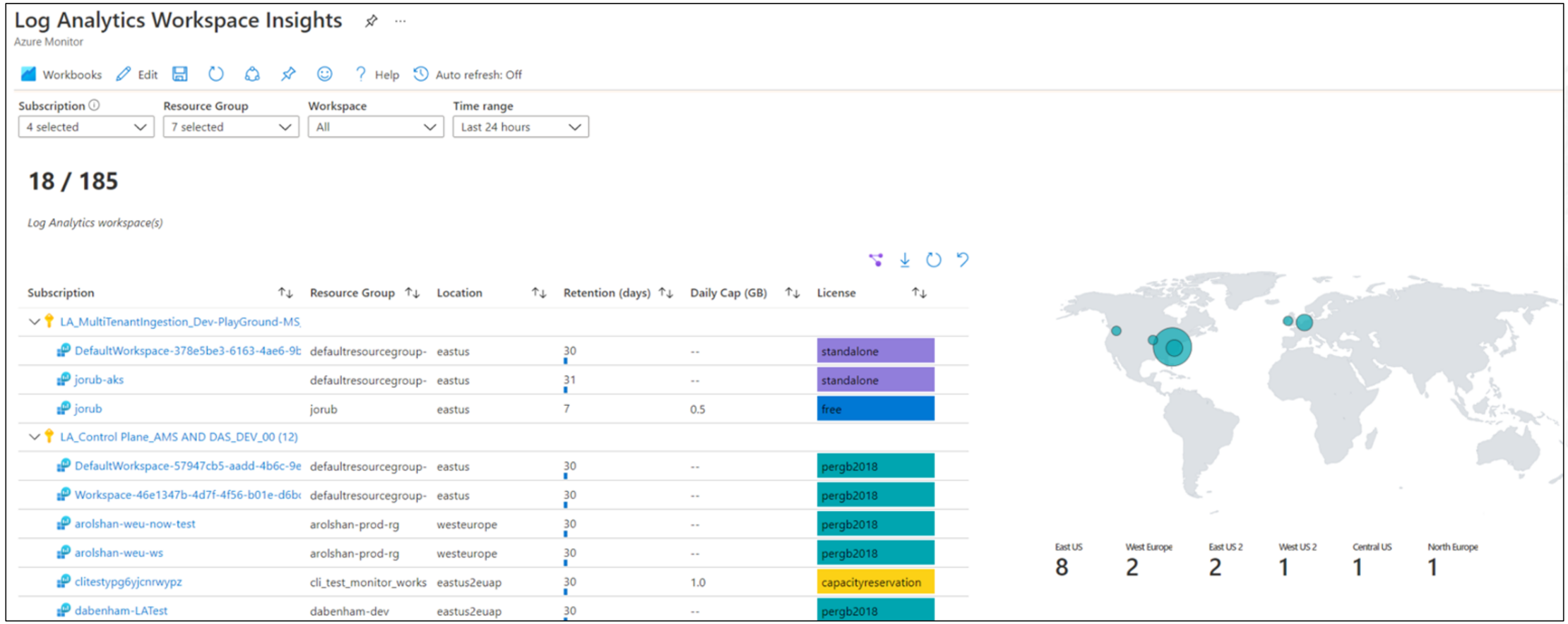
1. Find a pod’s Cluster IP
2. Find a service’s IP
3. Fetch PID of a container in Docker
4. Track connection

TECHNOLOGY

AKS Monitoring and Logging

Log Analytics Workspace

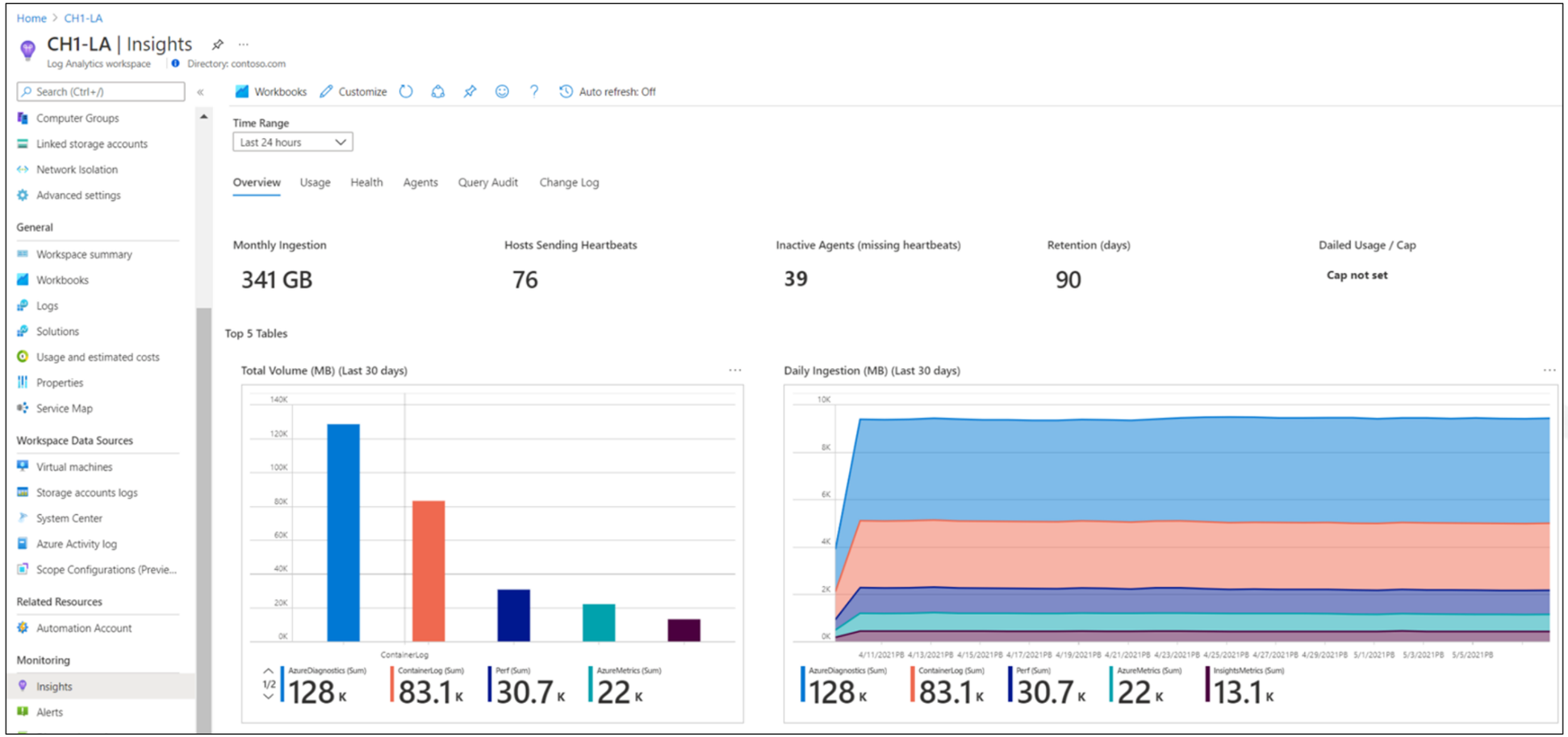
Log Analytics Workspace Insights (preview) provides comprehensive monitoring of your workspaces through a unified view of your workspace usage, performance, health, agent, queries, and change log.



Source: https://docs.microsoft.com/en-us/azure/azure-monitor/logs/log-analytics-workspace-insights-overview

Log Analytics Workspace

It helps manage multiple Clusters/Nodes and also monitor the performance of each of these components.



Source: https://docs.microsoft.com/en-us/azure/azure-monitor/logs/log-analytics-workspace-insights-overview

Azure Monitor

- Azure Monitor stores log data in a Log Analytics workspace. A workspace is a container that includes data and configuration information.
- Azure Monitor stores the data of all the functioning components of the AKS Cluster.
- It also provides an RBAC to limit access to sensitive data and maintain security.

AKS Monitoring and Logs

Problem Statement: Monitor health of an AKS cluster and check pod logs for troubleshooting.

ASSISTED PRACTICE

Assisted Practice: Guidelines

Steps to monitor and log AKS data:

1. Monitoring cluster health using Azure Monitor
2. Checking logs of a running pod using Azure Log Analytics
3. Checking logs of a running pod using Azure Cloud Shell

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 virtual machines or VMs led to slowing down technology and 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 Docker 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.
- Monitor logs from Pods, Replication Controller, or Services to troubleshoot the issues in the application.