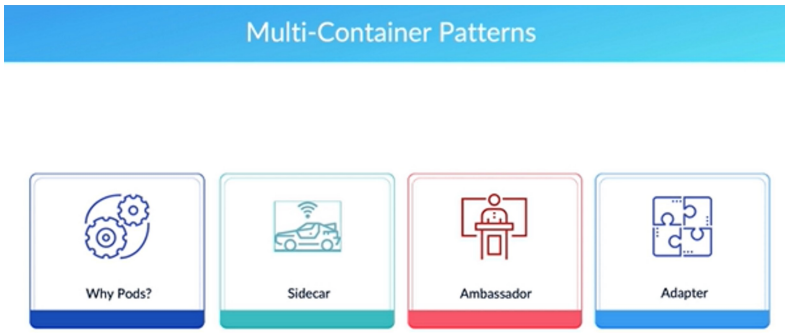
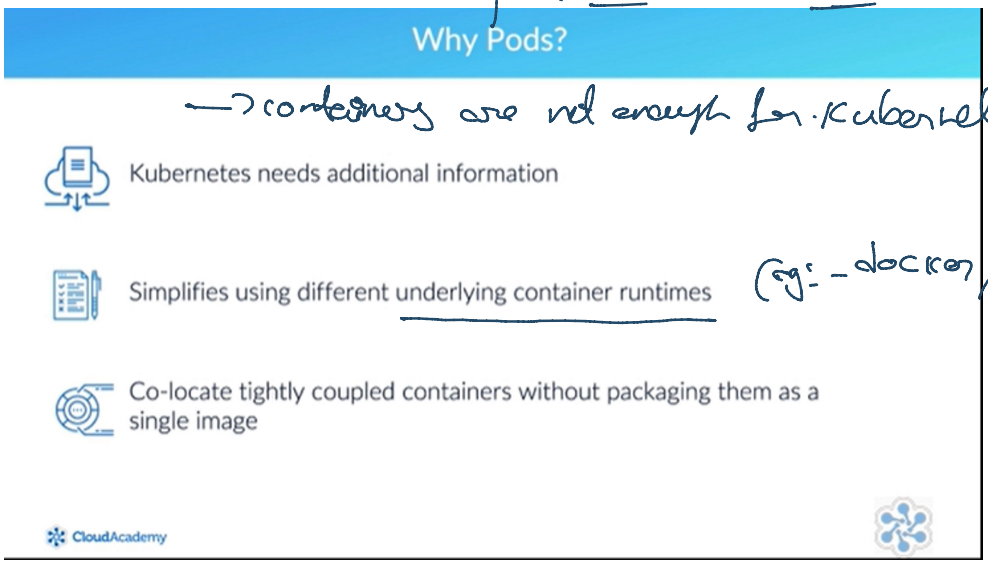
14. Multiple Container Patterns

01 March 2022 01:18



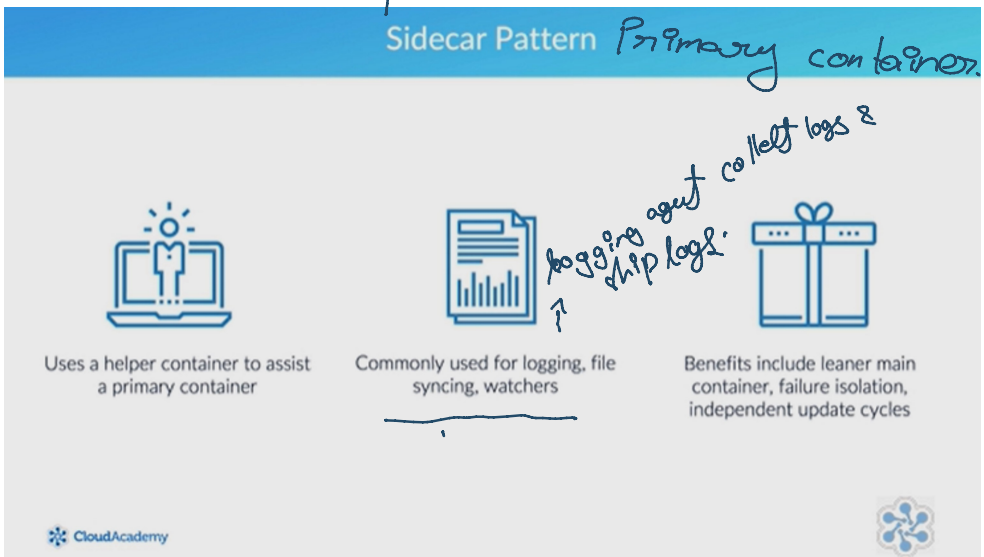
extra level of abstraction for containers.



→ containers are not enough for Kubernetes to effectively manage

(eg:- docker / rocket)

uses a helper container to assist the primary container



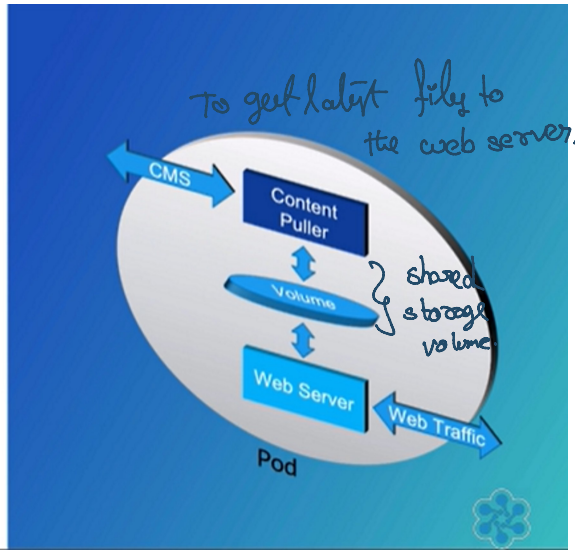
logging agent collect logs & ship logs.

all these can be accomplished by using a side car than using the main container to do these things.

Main container can do the traffic.

File Sync Sidecar Example

- Primary container: Web Server
- Sidecar: Content Puller
- Content Puller syncs with content management system (CMS)
- Web Server serves content
- The content is synced using a volume



CloudAcademy

Ambassador Pattern

Ambassador container is a proxy for communicating to and from the primary container

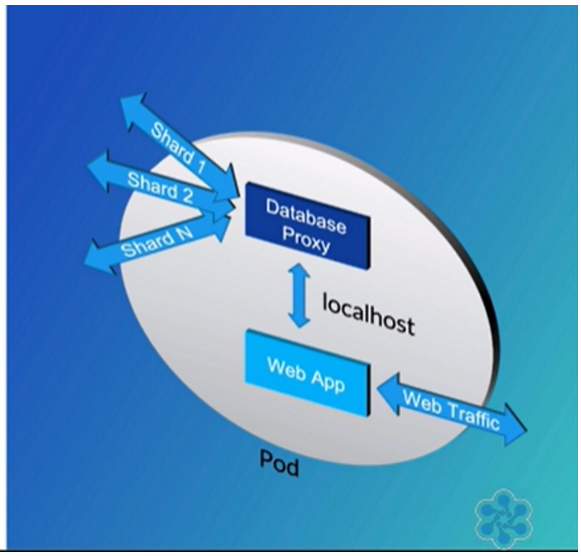
Commonly used for communicating with databases

Streamlined development experience, potential to reuse ambassador across languages

CloudAcademy

Database Ambassador Example

- Primary container: Web App
- Ambassador: Database Proxy
- Web App handles requests
- Database requests are sent to the Database Proxy over localhost
- Database Proxy then forwards the requests to the appropriate database
- Possibly sharding the requests



CloudAcademy

Adapter Pattern

Adapter Pattern

simple for outside world.

using a container

Adapters present a ~~standardized~~ interface across multiple Pods

Commonly used for normalizing output logs and monitoring data across the pods

Adapts third-party software to meet your needs

In this demonstration, we'll see how to implement the adapter pattern.

```
apiVersion: v1
kind: Pod
metadata:
  name: app
  namespace: legacy
spec:
  containers:
  - name: app
    image: alpine:3.9.2
    command: ["/bin/sh", "-c"]
    # Output legacy formatted metrics every 5 seconds
    args:
    - mkdir /metrics;
      while true; do
        date > /metrics/raw.txt;
        top -n 1 -b >> /metrics/raw.txt;
        sleep 5;
      done
~
~
~
~
~
~
~
~
"no-adapter.yaml" 18L, 367C
```

This app outputs raw metric data every 5 sec to a file.

↑ your monitoring solution require a JSON format which is a problem.

```
ubuntu@ip-10-0-128-5:~$ kubectl exec -n legacy app -- cat /metrics/raw.txt
Thu Mar 28 23:14:57 UTC 2019
Mem: 843844K used, 136912K free, 1180K shrd, 24920K buff, 410652K cached
CPU:   0% usr   0% sys   0% nic 100% idle   0% io   0% irq   0% sirq
Load average: 0.01 0.02 0.08 1/334 143
  PID  PPID USER     STAT   VSZ %VSZ CPU %CPU COMMAND
    1     0 root     S     1588   0%   0   0% /bin/sh -c mkdir /metrics; while true; do
 date > /metrics/raw.txt; top -n 1 -b >> /metrics/raw.txt; sleep 5; done
  143     1 root     R     1528   0%   0   0% top -n 1 -b

ubuntu@ip-10-0-128-5:~$
```

we only need data, memory & CPU that too in Json format

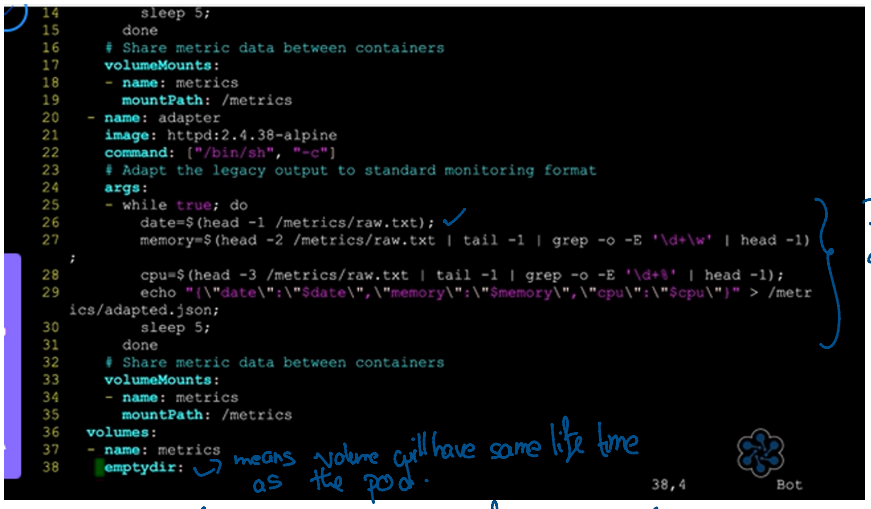
↳ This is where we can use adapter Pattern.

```
ubuntu@ip-10-0-128-5:~$ vi no-adapter.yaml
ubuntu@ip-10-0-128-5:~$
```

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: adapted-pod
spec:
  containers:
  - name: app
    image: alpine:3.9.2
    command: ["/bin/sh", "-c"]
    args:
    - while true; do
        date > /metrics/raw.txt;
        top -n 1 -b >> /metrics/raw.txt;
        sleep 5;
      done
    # Share metric data between containers
    volumeMounts:
    - name: metrics
      mountPath: /metrics
  - name: adapter
    image: httpd:2.4.38-alpine
    command: ["/bin/sh", "-c"]
    # Adapt the legacy output to standard monitoring format
    args:
    - while true; do
        date=$(head -1 /metrics/raw.txt);
@
"adapter.yaml" 38L, 1044C
```

} the legacy app remain same with one diff of volume so that raw metrics can be shared with adaptive container.

} → adaptive container,



} Json output every 5 seconds

↳ means volume will have same life time as the pod.

↳ there is no need of persistant volume.

```
ubuntu@ip-10-0-128-5:~$ kubectl create -f adapter.yaml
pod/adapted-pod created
ubuntu@ip-10-0-128-5:~$
```

A

```
ubuntu@ip-10-0-128-5:~$ kubectl exec adapted-pod --container=adapter -- cat /metrics/ad
apted.json
{"date":"Thu Mar 28 23:34:37 UTC 2019","memory":"839892K","cpu":"0%"}
ubuntu@ip-10-0-128-5:~$
```

} Json in the metrics

```
ubuntu@ip-10-0-128-5:~$ kubectl exec adapted-pod --container=adapter -- cat /metrics/ad
apted.json
{"date":"Thu Mar 28 23:34:42 UTC 2019","memory":"839620K","cpu":"0%"}
ubuntu@ip-10-0-128-5:~$
```

} We can use webserver to give the metric as a Rest API or configuring the adapter to Push Json to metrix aggregator system.

CONCLUSION

- Explained why Pods

CONCLUSION

- Explained why Pods
- Sidecar, ambassador, and adapter patterns
- Demo implementation of adapter pattern

Push json to metric
system.