Microsoft

# Modernizing Applications with Containers and Orchestrators

Microsoft Services

Microsoft

# Module 7 – Monitoring and Troubleshooting Containers

Microsoft Services

# Objectives

- Useful docker and kubectl commands for troubleshooting containers
- Azure Monitor for containers
- 3rd Party Azure Partner Solutions

# `docker commands` for troubleshooting

- docker logs
- docker stats
- docker attach
- docker top
- docker events
- docker inspect
- docker history

# docker logs

Retrieves container logs present at the time of execution

```
$ docker run --name test -d busybox sh -c "while true; do $(echo date); sleep 1; done"
$ date
Tue 14 Nov 2017 16:40:00 CET
$ docker logs -f --until=2s
Tue 14 Nov 2017 16:40:00 CET
Tue 14 Nov 2017 16:40:01 CET
Tue 14 Nov 2017 16:40:02 CET
```

*Retrieve logs until a specific point in time

```
docker cp <container_id>:/path/to/useful/file /local-path
```

# docker stats

A live stream of resource usage, so you can see just how much memory you've leaked so far

```
$ docker stats

CONTAINER ID        NAME                                CPU %               MEM USAGE / LIMIT     MEM %               NET I/O             BLOCK I/O           PIDS
b95a83497c91        awesome_brattain                    0.28%               5.629MiB / 1.952GiB   0.28%               916B / 0B           147kB / 0B          9
67b2525d8ad1        foobar                              0.00%               1.727MiB / 1.952GiB   0.09%               2.48kB / 0B         4.11MB / 0B         2
e5c383697914        test-1951.1.kay7x1lh1twk9c0oig50sd5tr   0.00%               196KiB / 1.952GiB     0.01%               71.2kB / 0B         770kB / 0B          1
4bda148efbc0        random.1.vnc8on831idyr42slu578u3cr      0.00%               1.672MiB / 1.952GiB   0.08%               110kB / 0B          578kB / 0B          2
```

# Stream stdout with `docker attach`

Attach your terminal's standard input, output, and error (or any combination of the three) to a running container using the container's ID or name.

# Get process stats with `docker top`

Displays all running processes of a container including information about process name, ID, CPU usage, and private memory

# docker events

Docker containers will report the following real-time events: create, destroy, die, export, kill, oom, pause, restart, start, stop, and unpause

```
$ docker events --filter 'event=stop'

2017-01-05T00:40:22.880175420+08:00 container stop 0fdb...ff37 (image=alpine:latest, name=test)
2017-01-05T00:41:17.888104182+08:00 container stop 2a8f...4e78 (image=alpine, name=kickass_brattain)

$ docker events --filter 'image=alpine'

2017-01-05T00:41:55.784240236+08:00 container create d9cd...4d70 (image=alpine, name=happy_meitner)
2017-01-05T00:41:55.913156783+08:00 container start d9cd...4d70 (image=alpine, name=happy_meitner)
2017-01-05T00:42:01.106875249+08:00 container kill d9cd...4d70 (image=alpine, name=happy_meitner, signal=15)
2017-01-05T00:42:11.111934041+08:00 container kill d9cd...4d70 (image=alpine, name=happy_meitner, signal=9)
2017-01-05T00:42:11.119578204+08:00 container die d9cd...4d70 (exitCode=137, image=alpine, name=happy_meitner)
2017-01-05T00:42:11.173276611+08:00 container stop d9cd...4d70 (image=alpine, name=happy_meitner)

$ docker events --filter 'container=test'

2017-01-05T00:43:00.139719934+08:00 container start 0fdb...ff37 (image=alpine:latest, name=test)
2017-01-05T00:43:09.259951086+08:00 container kill 0fdb...ff37 (image=alpine:latest, name=test, signal=15)
2017-01-05T00:43:09.270102715+08:00 container die 0fdb...ff37 (exitCode=143, image=alpine:latest, name=test)
2017-01-05T00:43:09.312556440+08:00 container stop 0fdb...ff37 (image=alpine:latest, name=test)
```

attach
commit
copy
create
destroy
detach
die
exec_create
exec_detach
exec_die
exec_start
export
health_status
kill
oom
pause
rename
resize
restart
start
stop
top
unpause
update

# View container details with `docker inspect`

- Return low-level information on Docker objects
- By default, docker inspect will render results in a JSON format which can be parsed inline or saved on file

### Get an instance's IP address

For the most part, you can pick out any field from the JSON in a fairly straightforward manner.

```
$ docker inspect --format='{{range .NetworkSettings.Networks}}{{.IPAddress}}{{end}}' $INSTANCE_ID
```

### Get an instance's MAC address

```
$ docker inspect --format='{{range .NetworkSettings.Networks}}{{.MacAddress}}{{end}}' $INSTANCE_ID
```

### Get an instance's log path

```
$ docker inspect --format='{{.LogPath}}' $INSTANCE_ID
```

# docker history

- Provides full history of an image
- Useful for retrieving full footprint of each layer and subsequent sizes etc.

```
C:\>docker history mcr.microsoft.com/dotnet/core/aspnet:2.2
IMAGE               CREATED             CREATED BY                                      SIZE
5f58a78e0e06        2 weeks ago         /bin/sh -c curl -SL --output aspnetcore.tar.…   154MB
<missing>           2 weeks ago         /bin/sh -c #(nop)  ENV ASPNETCORE_VERSION=2.…   0B
<missing>           2 weeks ago         /bin/sh -c apt-get update     && apt-get ins…   7.02MB
<missing>           2 weeks ago         /bin/sh -c #(nop)  ENV ASPNETCORE_URLS=http:…   0B
<missing>           2 weeks ago         /bin/sh -c apt-get update     && apt-get ins…   43.8MB
<missing>           2 weeks ago         /bin/sh -c #(nop)  CMD ["bash"]                 0B
<missing>           2 weeks ago         /bin/sh -c #(nop) ADD file:4fc310c0cb879c876…   55.3MB
```

# Can’t start your container at all?

1. Save the current state of the shut-down container as a new image
2. Start that with a different command to avoid your existing failures

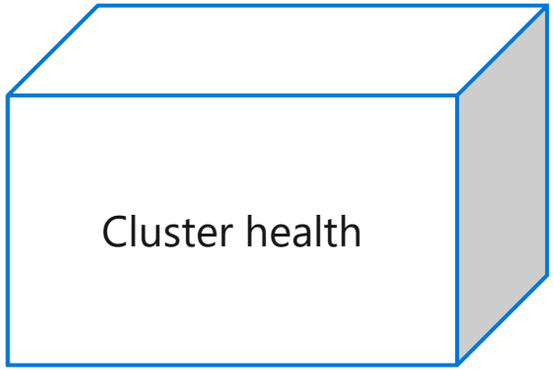
```
docker commit <container_id> my-broken-container &&
docker run -it my-broken-container /bin/bash
```

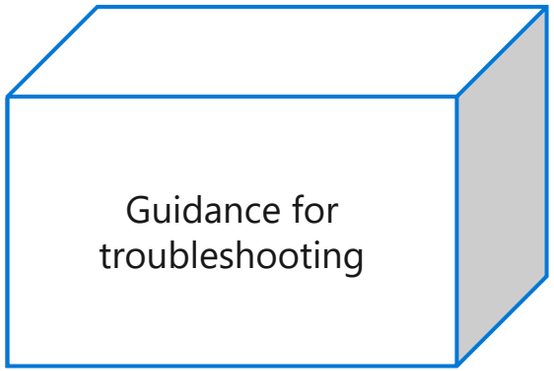
# kubectl troubleshooting commands

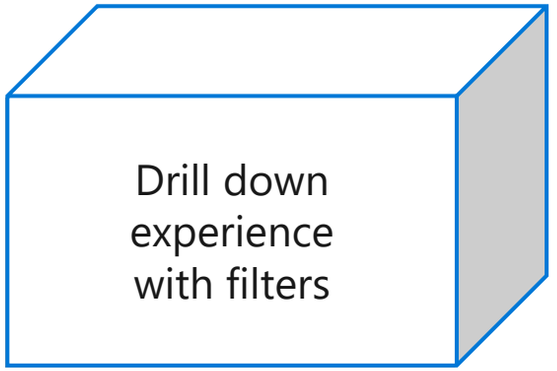
- `kubectl logs my-pod`
- `kubectl describe pods my-pod`
- `kubectl exec my-pod -- <your command>`
  - Will run the command inside the container
  - Ex. kubectl exec -it my-pod -- Powershell

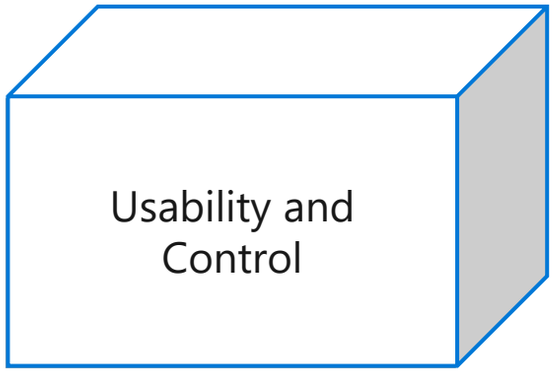
https://kubernetes.io/docs/tasks/debug-application-cluster/debug-pod-replication-controller/

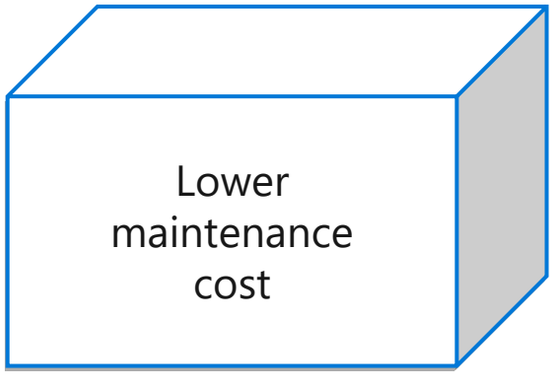
# Common container monitoring pain points…

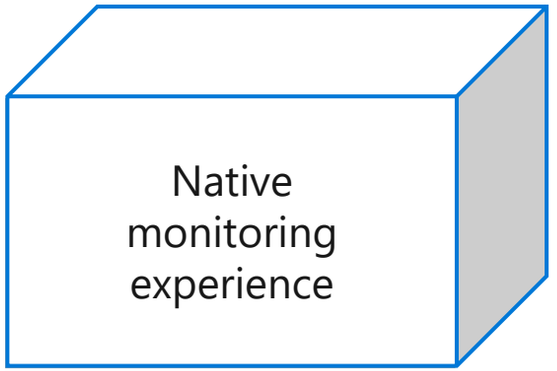


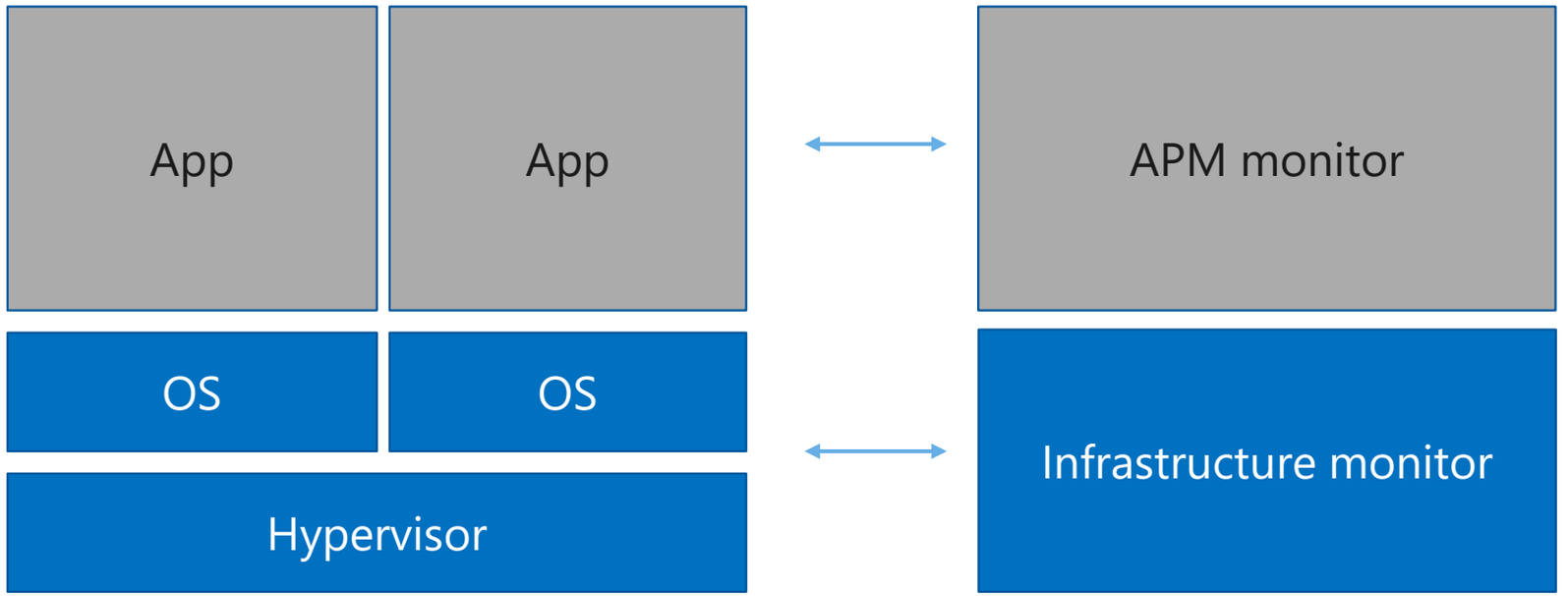



# Things to monitor

- **Cluster health**: general status, number of nodes, pods
- **Node health**: available compute resources, status,…
- **App health**: the application is working properly?

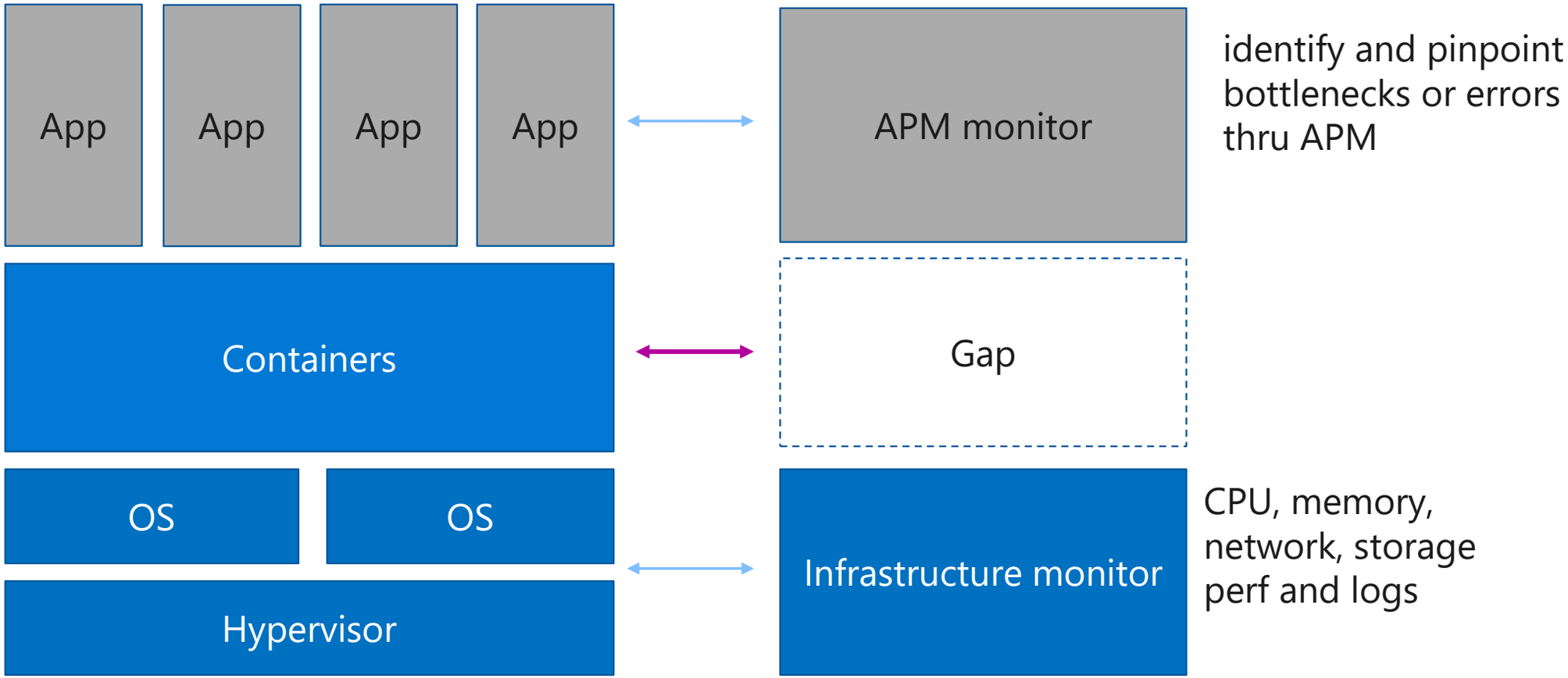
# Traditional Monitoring



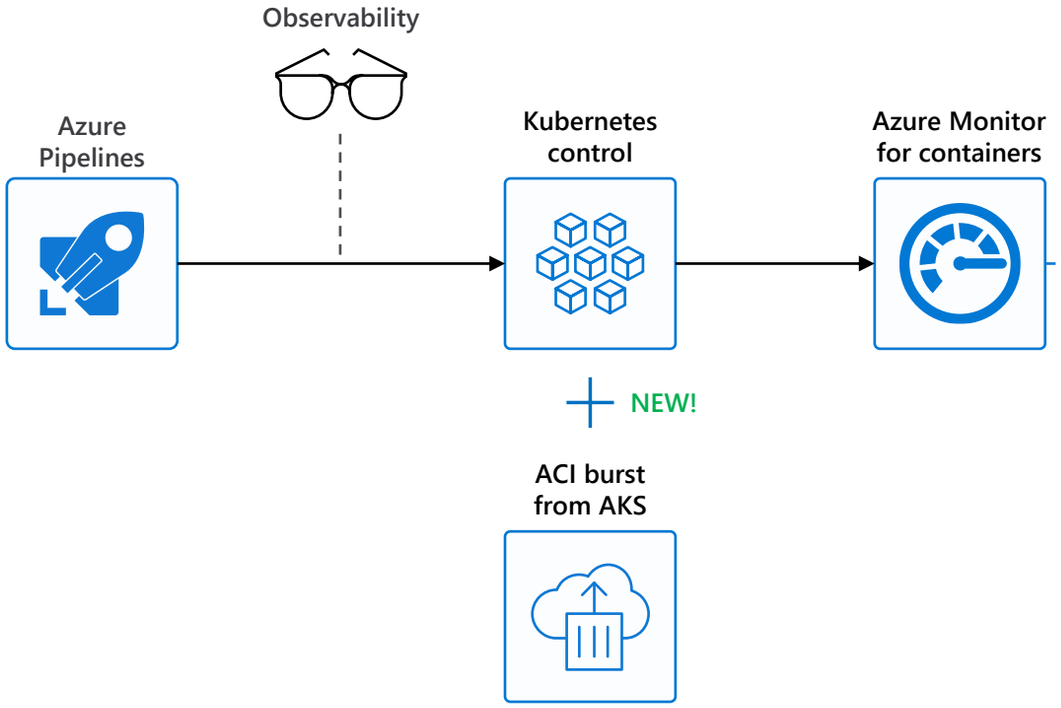

identify and
pinpoint bottlenecks
or errors
thru APM

CPU, memory,
network, storage
perf and logs

# Gap in Stack with Containers



# Azure Monitor for Containers overview




**Visualization**
Visualize overall health and performance from clusters to containers with drill downs and filters

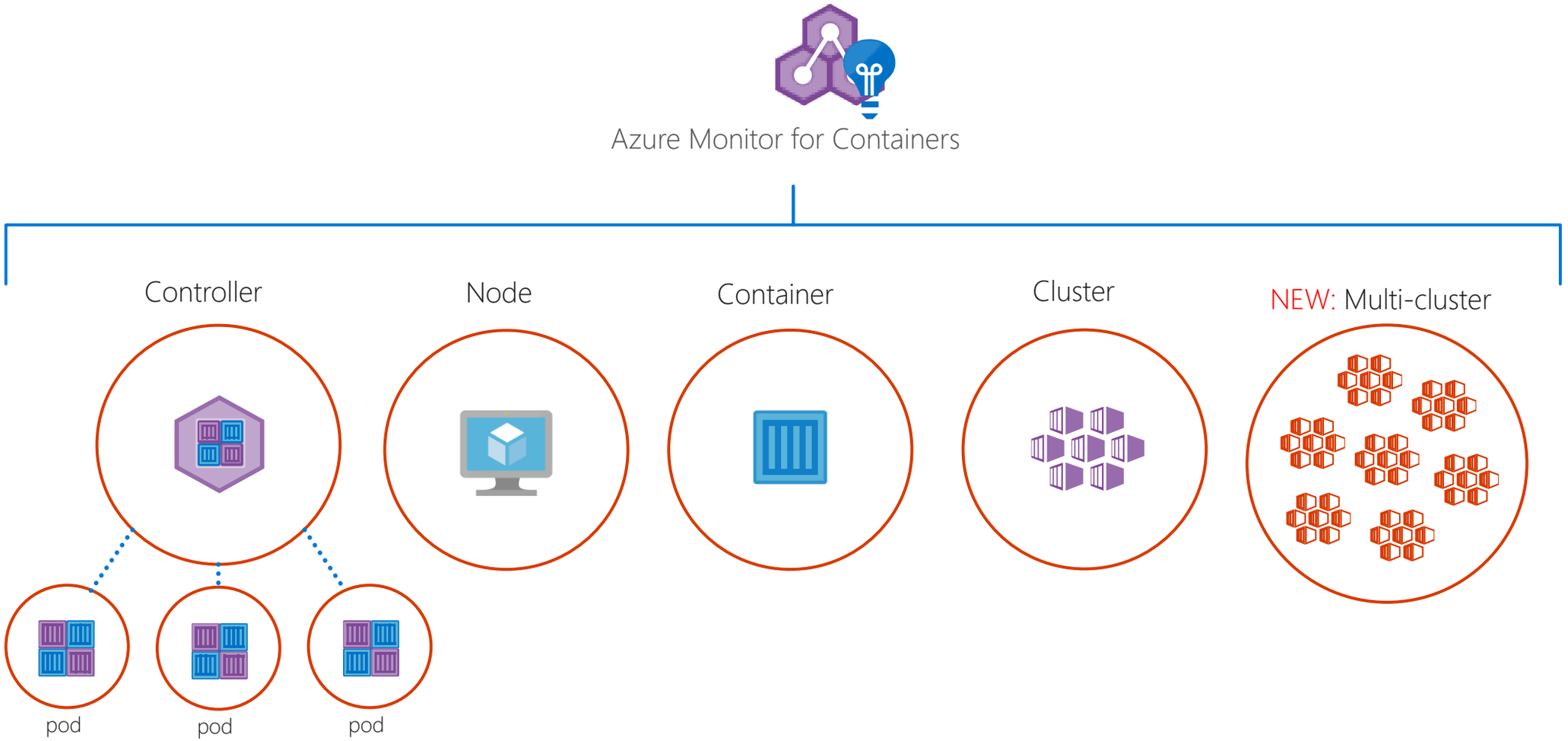
**Insights**
Provide insights with multi-cluster health roll up view

**Monitor & Analyze**
Monitor and analyze Kubernetes and container deployment performance, events, health, and logs

**Response**
Native alerting with integration to issue managements and ITSM tools

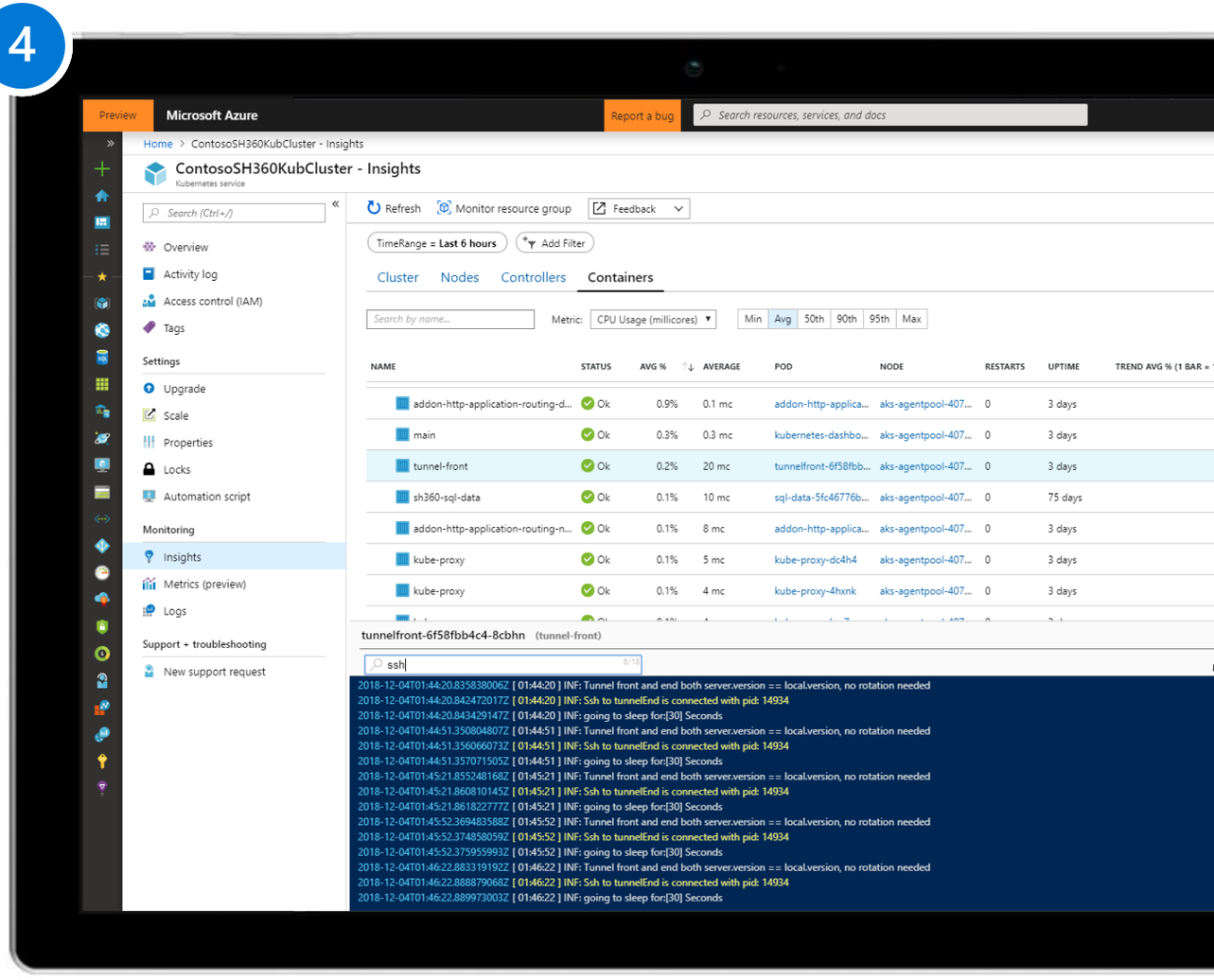
**Observability**
Observe live container logs on container deployment status

# Single-pane view



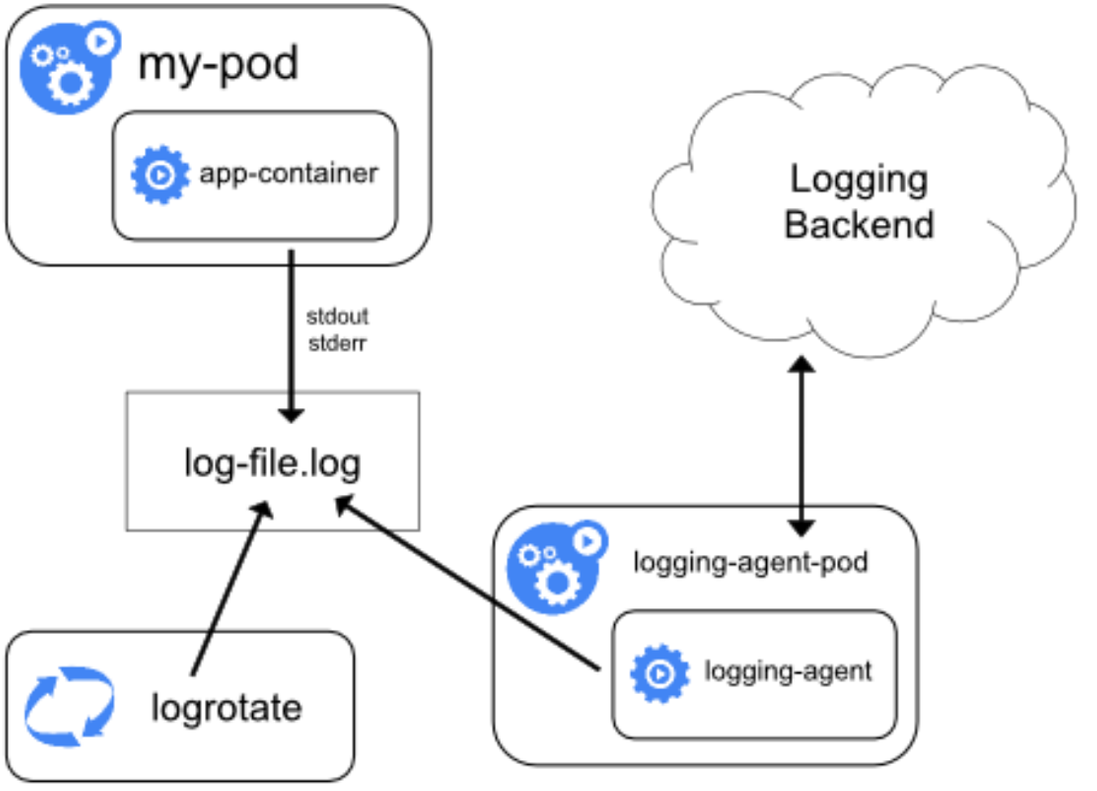
# Azure Monitor for Containers UI

1. Get detailed insights about your workloads with Azure Monitor
2. See graphical insights about clusters
3. Filter for details about nodes, controllers, and containers
4. Pull events and logs for detailed activity analysis



# Logging architecture – node-level logging agent

- Most common and encouraged approach for Kubernetes cluster
- One agent per node (Daemon Set)
- No changes to the applications running on the node
- Only works for applications with standard output and standard error



# Enable monitoring - Portal

## Create Kubernetes cluster

Basics | Authentication | Networking | Monitoring | Tags | Review + create

With Azure Kubernetes Service, you will get CPU and memory usage metrics for each node. In addition, you can enable container monitoring capabilities and get insights into the performance and health of your entire Kubernetes cluster. You will be billed based on the amount of data ingested and your data retention settings.

Learn more about container performance and health monitoring
Learn more about pricing

**AZURE MONITOR**

Enable container monitoring: No | Yes

Log Analytics workspace: DefaultWorkspace-ed29c799-3b06-4306-971a-202c3c2d29a9-WEU

Create new

# Enable monitoring - CLI

**Creating new AKS Cluster**

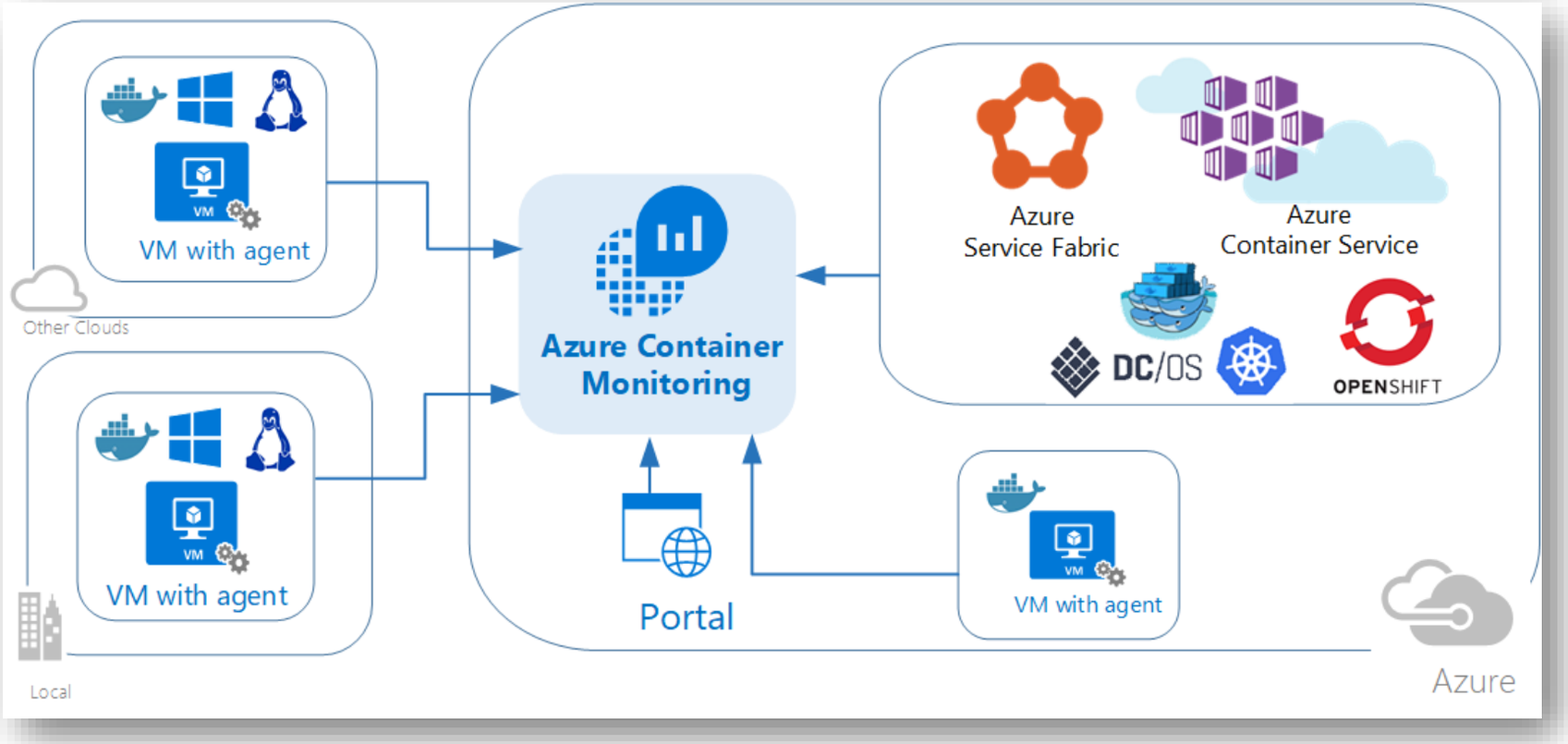
az aks create --resource-group myAKSCluster --name myAKSCluster --node-count 1 **--enable-addons monitoring** --generate-ssh-keys

**Enable monitoring on existing AKS Cluster**

az aks **enable-addons -a monitoring** -n MyExistingManagedCluster -g MyExistingManagedClusterRG

# Container Monitoring solution in Azure Monitor

- View and manage your Docker and Windows container hosts in a single location
- The solution shows which containers are running, what container image they're running, and where containers are running




Supports:
Docker Swarm
DC/OS
Kubernetes
Service Fabric
Red Hat OpenShift

# Other monitoring options

splunk>

Prometheus

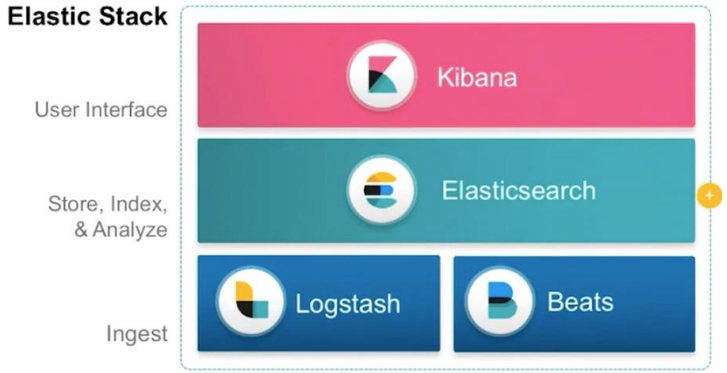
fluentd

DATADOG

Grafana

COSCALE

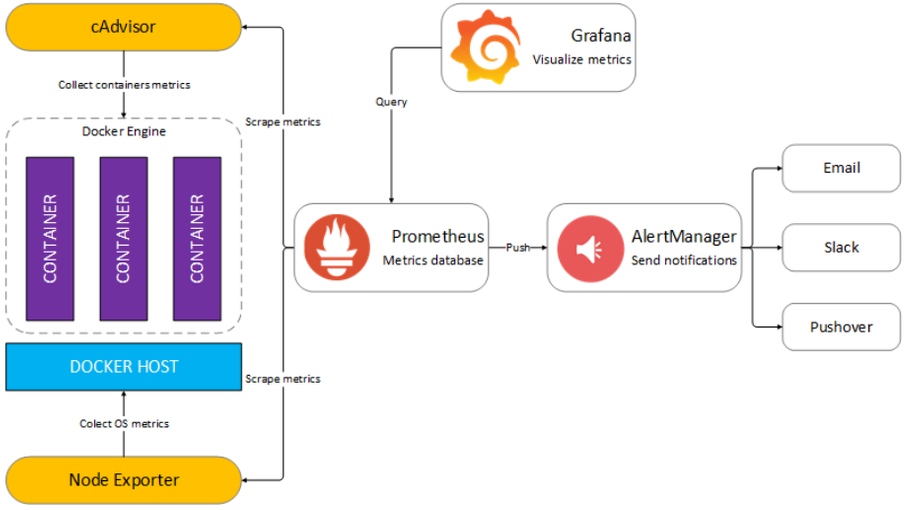
sysdig




OPENTRACING

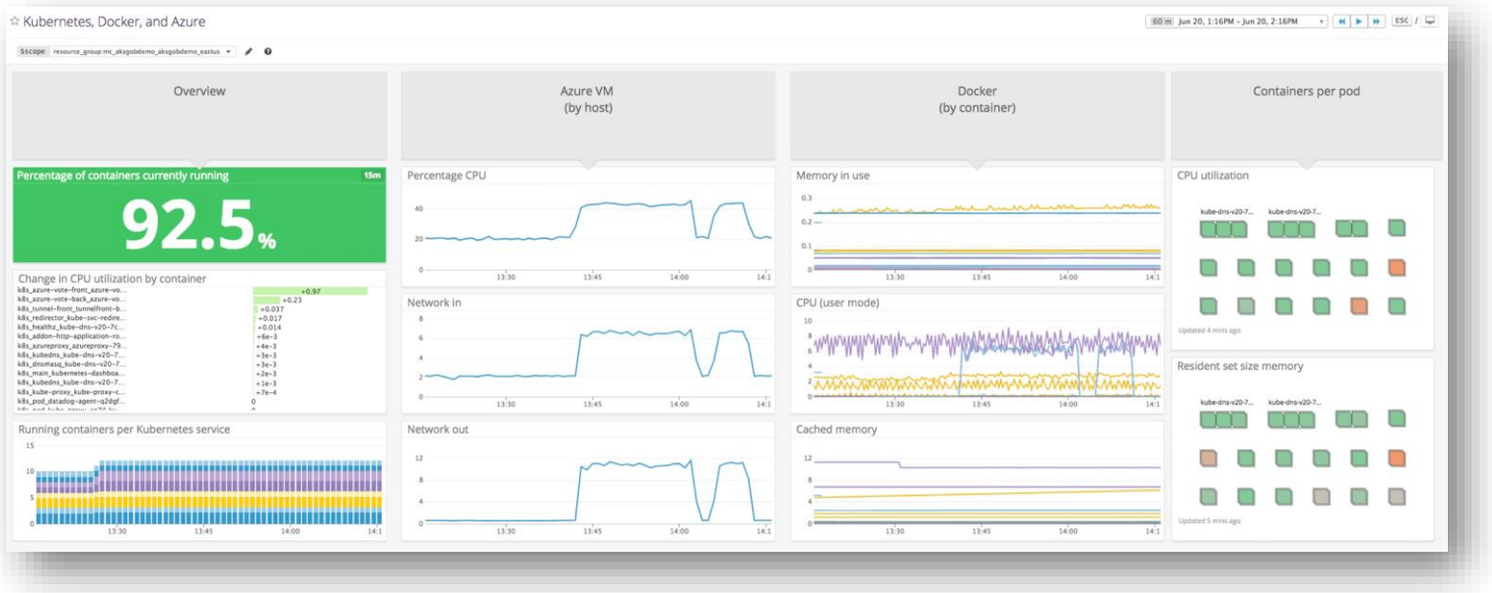
# Prometheus + Grafana

- Prometheus is an open-source systems monitoring and alerting toolkit
- Grafana allows you to query, visualize, alert on and understand your metrics no matter where they are stored.



# Datadog

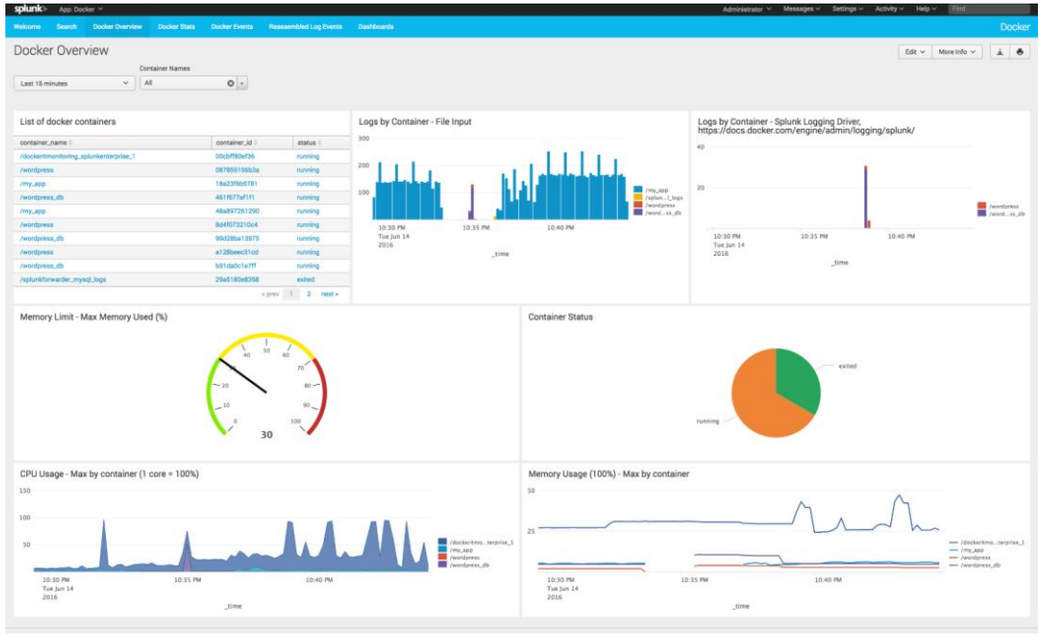
- Datadog's integrations with Kubernetes and Azure Monitor provide comprehensive visibility into AKS infrastructure with no additional configuration
- To use, you deploy the containerized Datadog Agent as a DaemonSet within your AKS cluster using a provided YAML manifest

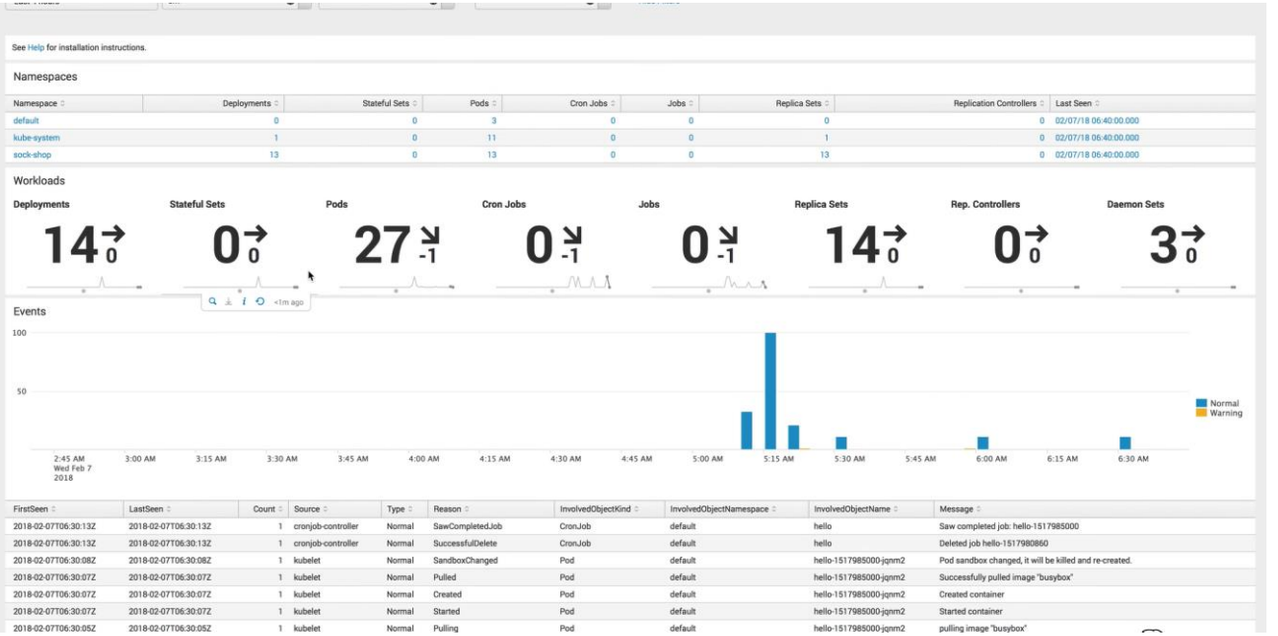



https://www.datadoghq.com/blog/announcing-aks/

# Splunk

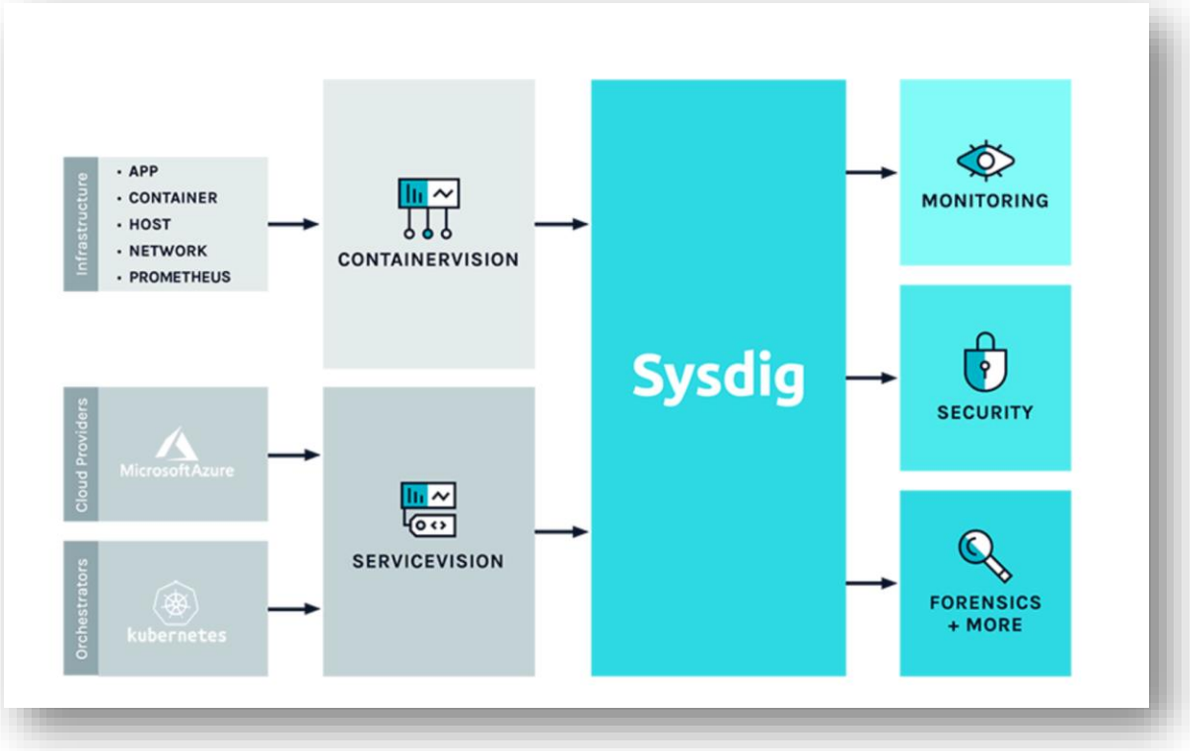
Collect, analyze and act upon the untapped value of the big data generated by your technology infrastructure, security systems and business applications giving you the insights to drive operational performance and business results

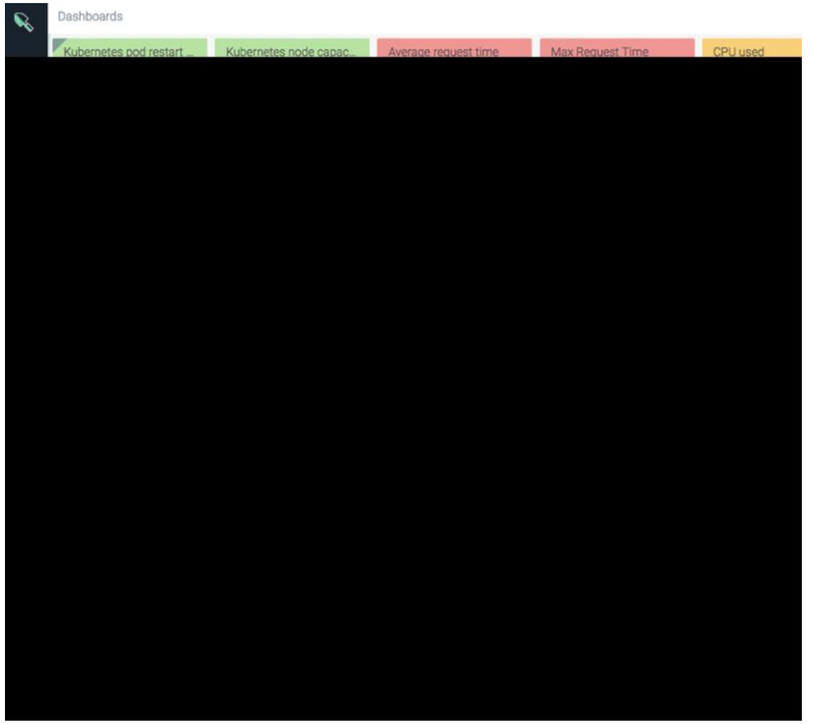


# Sysdig

- Monitoring and troubleshooting service provides dashboard based on various metrics like CPU, Memory, Network etc.
- Service-level visibility provided by leveraging metadata from AKS




https://sysdig.com/partners/microsoft-azure/

© 2015 Microsoft Corporation. All rights reserved.