REFERENCE ARCHITECTURE – June 2023

Running Flink and K8ssandra on VMware Tanzu Kubernetes Grid 2

Contents

Executive Summary

Many organizations generate vast amounts of data in short intervals from various sources like sensors, user activity, and transactional systems. There is a growing need to process this streaming data in real-time to gain meaningful insights and make critical business decisions. Stream computing can help organizations achieve this by instantaneously processing insights, storing them immutably, scaling horizontally, and ensuring high availability, which enables innovative applications and real-time aligned business outcomes.

Apache Flink is a stream processing framework that helps solve this problem. Flink processes enormous volumes of data in real-time, generating insights, alerts, and dashboards instantly. Meanwhile, Apache Cassandra is a distributed database that can store large amounts of data and scale elastically, providing an immutable audit trail. They scale horizontally to handle petabyte workloads and peak throughput demands, with inherent fault tolerance features.

Deploying and managing these streaming applications at scale can be challenging. Using VMware Tanzu® Kubernetes Grid™ to run and manage Apache Flink and Cassandra can provide organizations with a scalable, reliable, and manageable solution to process and store large volumes of streaming data in real-time.

This paper demonstrates how Flink and Cassandra integrate seamlessly on Tanzu Kubernetes Grid to build highly scalable real-time streaming analytics applications with robust processing, availability, scalability, and governance. Real-time insights and immutable storage meet scalability and consistency at scale, enabling advanced use cases such as IoT and log processing, recommendations, and reporting. Enterprises can thus realize the potential of streaming data through Flink and Cassandra to disrupt markets. Running both platforms on Tanzu Kubernetes Grid delivers a robust and scalable platform for streaming analysis.

Technology Overview

The technological components in this solution are:

- VMware Tanzu Kubernetes Grid 2
- Apache Flink
- K8ssandra
- Monitoring tools

VMware Tanzu Kubernetes Grid

VMware Tanzu Kubernetes Grid provides organizations with a consistent, upstream-compatible Kubernetes substrate that is ready for end-user workloads and ecosystem integrations. It uses a new API called ClusterClass that defines a common cluster configuration for different infrastructure providers. Tanzu Kubernetes Grid 2 deploys clusters using an opinionated configuration of Kubernetes open-source software that is supported by VMware, so that you do not have to build a Kubernetes environment by yourself, it also provides packaged services such as networking, authentication, ingress control, and logging that are required for production Kubernetes environments.

Tanzu Kubernetes Grid 2 supports two types of deployment models: Supervisor deployment and standalone management cluster deployment. Supervisor deployment allows you to create and operate workload clusters natively in VMware vSphere® with Tanzu and leverage vSphere features. Standalone management cluster deployment allows you to create workload clusters on vSphere 6.7, 7, and 8 without Supervisor, or on AWS.

Flink

Apache Flink is a framework for distributed stream processing. It can process data in real-time streams and can also run batch processing jobs. Flink uses operator-based APIs that provide reusable data processing building blocks. Some common Flink operators include:

- Data sources: Read data from sources like Kafka, Kinesis, and files.
- Transformations: Map, filter, aggregate, and join to transform data
- Data sinks: Write output to sinks like Kafka, Cassandra, and ElasticSearch

Users can compose operators into data processing pipelines and Flink manages distributed processing, fault tolerance, state management, and optimization. Flink powers real-time data applications, and the operator-based APIs allow for declarative and scalable data processing.

Flink Kubernetes Operator allows deploying Flink on Kubernetes. It handles provisioning and lifecycle management of Flink clusters on Kubernetes. This allows running Flink easily and efficiently on cloud-native Kubernetes infrastructures. See https://flink.apache.org for more information.

K8ssandra

K8ssandra is a cloud-native distribution of Apache Cassandra® that runs on Kubernetes. K8ssandra provides an ecosystem of tools to provide richer data APIs and automated operations alongside Cassandra. This includes metric monitoring to promote observability, data anti-entropy services to support reliability, and backup/restore tools to support high availability and disaster recovery. As an open-source project licensed under Apache Software License version 2, K8ssandra is free to use, improve, and enjoy. K8ssandra integrates and packages together:

- Apache Cassandra
- Stargate, the open-source data gateway
- Cass-operator, the Kubernetes Operator for Apache Cassandra
- Reaper for Apache Cassandra, an anti-entropy repair feature (plus reaper-operator)
- Medusa for Apache Cassandra for backup and restore (plus medusa-operator)
- Metrics Collector for Apache Cassandra, with Prometheus integration, and visualization via pre-configured Grafana dashboards

See the k8ssandra website for more information.

Monitoring Tools

Prometheus is an open-source monitoring and alerting toolkit that is commonly used for Kubernetes but also supports other cloud-native environments. It is a high-scalable open-source monitoring framework that provides out-of-the-box monitoring capabilities for the Kubernetes container orchestration platform.

Grafana is an open-source analytics and monitoring platform that is commonly used for Kubernetes but also supports other cloud-native environments. It provides a powerful and elegant way to create, explore, and share dashboards and data with your team and others. Grafana can be used with Prometheus as a data source to visualize the metrics collected by Prometheus. Grafana can also be used with other data sources such as Graphite, Elasticsearch, Influx DB, and more.

vSAN Performance Service is used to monitor the performance of the vSAN environment, using the vSphere web client. The performance service collects and analyzes performance statistics and displays the data in a graphical format. You can use the performance charts to manage your workload and determine the root cause of problems.

Solution Configuration

Architecture

Two Tanzu workload clusters, each with multiple worker nodes, are provisioned. Persistent volumes are backed by vSAN. The k8ssandra operator is installed in cluster 1, and the Flink Operator is installed on cluster 2.

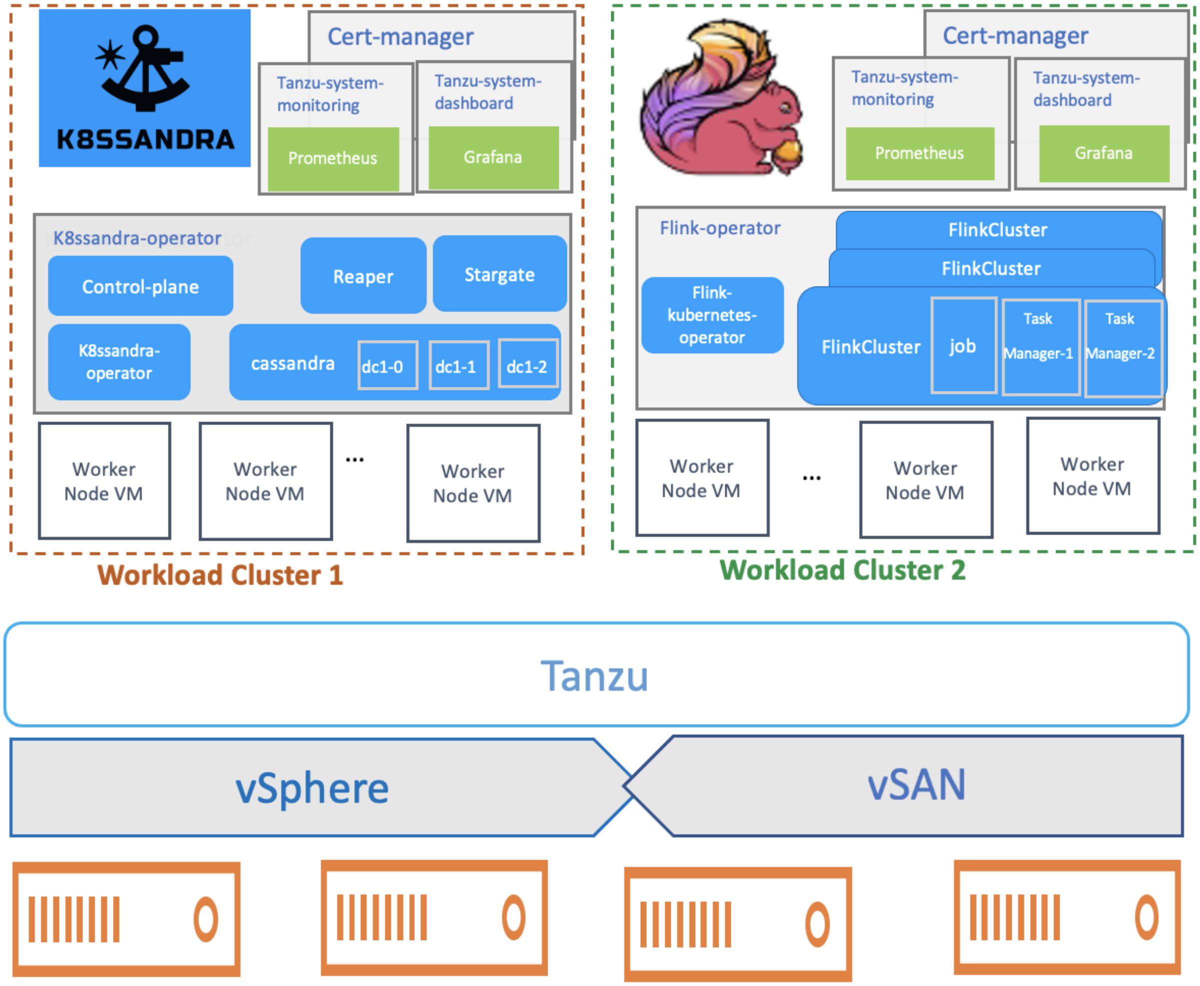


Figure 1. Solution Architecture

Monitoring service Prometheus is deployed in namespace tanzu-system-monitoring and Grafana is deployed in namespace tanzu-system-dashboard.

Note: Our validation is based on vSphere 8 cluster.

Hardware Resource

Table 1. Hardware Resource

Property	Specification
Server model name	4 x Dell PowerEdge R640
CPU	Intel(R) Xeon(R) Gold 6132 CPU @ 2.60GHz, 28 cores each
RAM	512GB
Network adapter	2 x Intel(R) Ethernet Controller 10G X550 2 x Intel Corporation I350 Gigabit Network Connection
Storage adapter	1 x Dell HBA330 Mini 2 x Express Flash PM1725a 1.6TB AIC
Disks	Cache - 2 x 1.6TB Dell Express Flash NVMe PCIe SSD PM1725a Capacity - 8 x 1.92TB write-intensive SAS SSDs

Software Resource

Table 2. Software Resource

Software	Version
vSphere	8.0
Tanzu Kubernetes Release	v1.21.6+vmware.1
K8ssandra	K8ssandra-operator v1.5.2
Flink	Flink operator v1.3.1
MetalLB	MetalLB v0.13.9

Network Configuration

Tanzu Kubernetes Grid is configured with Virtual Distributed Scheduler (VDS). MetalLB (see *installation* instructions) provides load balancing for external services. The Cassandra pods are placed in a namespace, and the Stargate and Reaper services can also be configured as load balancers. The configuration YAML files are *here*.

Tanzu Kubernetes Grid Configuration

Tanzu Kubernetes Grid uses a management cluster to create and manage workload clusters and has different deployment options based on the location that management cluster runs.

We followed Install the Tanzu CLI and Other Tools for Use with Standalone Management Clusters for the installation and deployed Management Clusters with the Installer Interface.

Then we deployed Tanzu Kubernetes Grid 2.1 with a standalone management cluster on vSphere 8 without a supervisor (see the guide).

Creating Workload Clusters

In VMware Tanzu Kubernetes Grid, workload clusters are the Kubernetes clusters on which your application workloads run. We followed the instruction that defined the workload cluster with Class-based clusters control planes and worker nodes as follows.

Table 3. Tanzu Kubernetes Cluster Definition

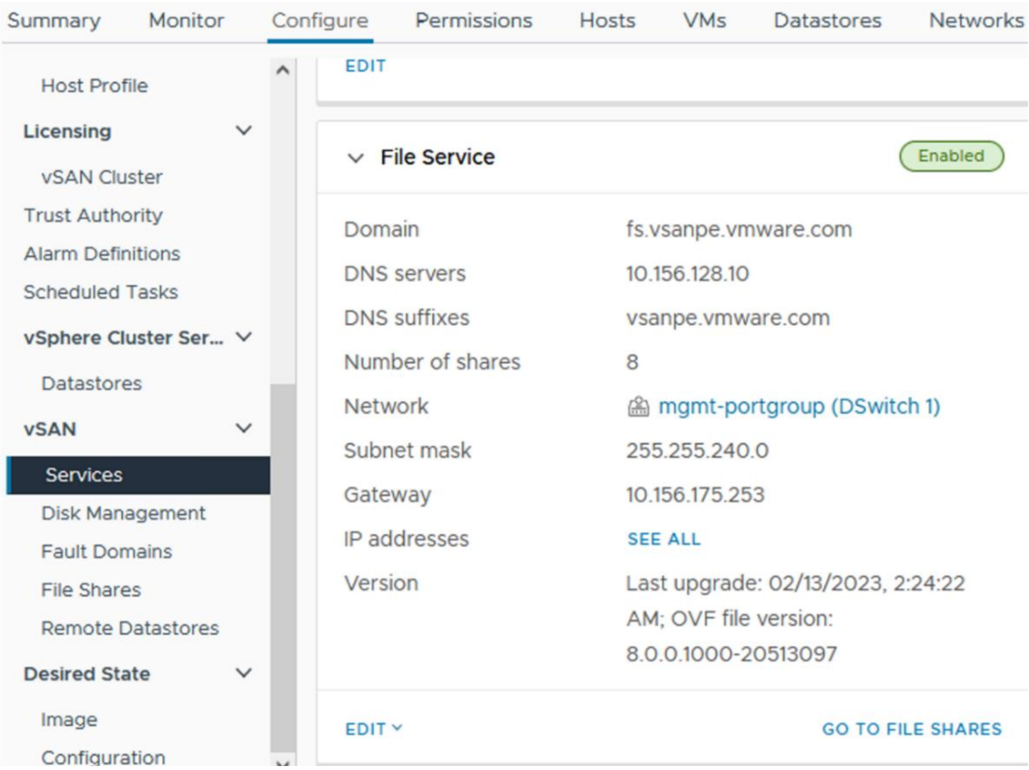
Role	Replicas	VM Configuration	Tanzu Kubernetes Release (TKR)
Control Plane	3	best-effort-small machine: diskGiB: 40 memoryMiB: 8192 numCPUs: 2	v1.24.9+vmware.1-tkg.1
Worker Nodes	3	best-effort-2xlarge machine: diskGiB: 40 memoryMiB: 65536 numCPUs: 16	v1.24.9+vmware.1-tkg.1

Role	Replicas	VM Configuration	Tanzu Kubernetes Release (TKR)
Control Plane	3	best-effort-small machine: diskGiB: 40 memoryMiB: 8192 numCPUs: 2	v1.24.9+vmware.1-tkg.1
Worker Nodes	8 (initially 3, later scaled)	best-effort-2xlarge machine: diskGiB: 40 memoryMiB: 16192 numCPUs: 4	v1.24.9+vmware.1-tkg.1

The YAML files used in our validation for workload cluster deployment can be found *here.*

ReadWriteMany File Volume Configuration

File volumes backed by vSAN file shares can be created statically or dynamically and mounted by stateful containerized applications. See the guide to enable file volumes.



Domain	fs.vsanpe.vmware.com
DNS servers	10.156.128.10
DNS suffixes	vsanpe.vmware.com
Number of shares	8
Network	mgmt-portgroup (DSwitch 1)
Subnet mask	255.255.240.0
Gateway	10.156.175.253
IP addresses	SEE ALL
Version	Last upgrade: 02/13/2023, 2:24:22 AM; OVF file version: 8.0.0.1000-20513097

Figure 2. File Volume Configuration

Note: The Tanzu Kubernetes Grid cluster using Tanzu Kubernetes Grid role needs to assign the required privilege**: Host.Configuration.Storage partition configuration**, see *vSphere Roles and Privileges* for more information.

Prometheus and Grafana Deployment

Tanzu Kubernetes Grid provides cluster monitoring services by implementing the open-source Prometheus and Grafana projects.
By using Prometheus and Grafana, you can gain insights into the health and performance of your Tanzu Kubernetes Grid clusters. This information can help you identify and troubleshoot problems and ensure that your clusters are running smoothly.

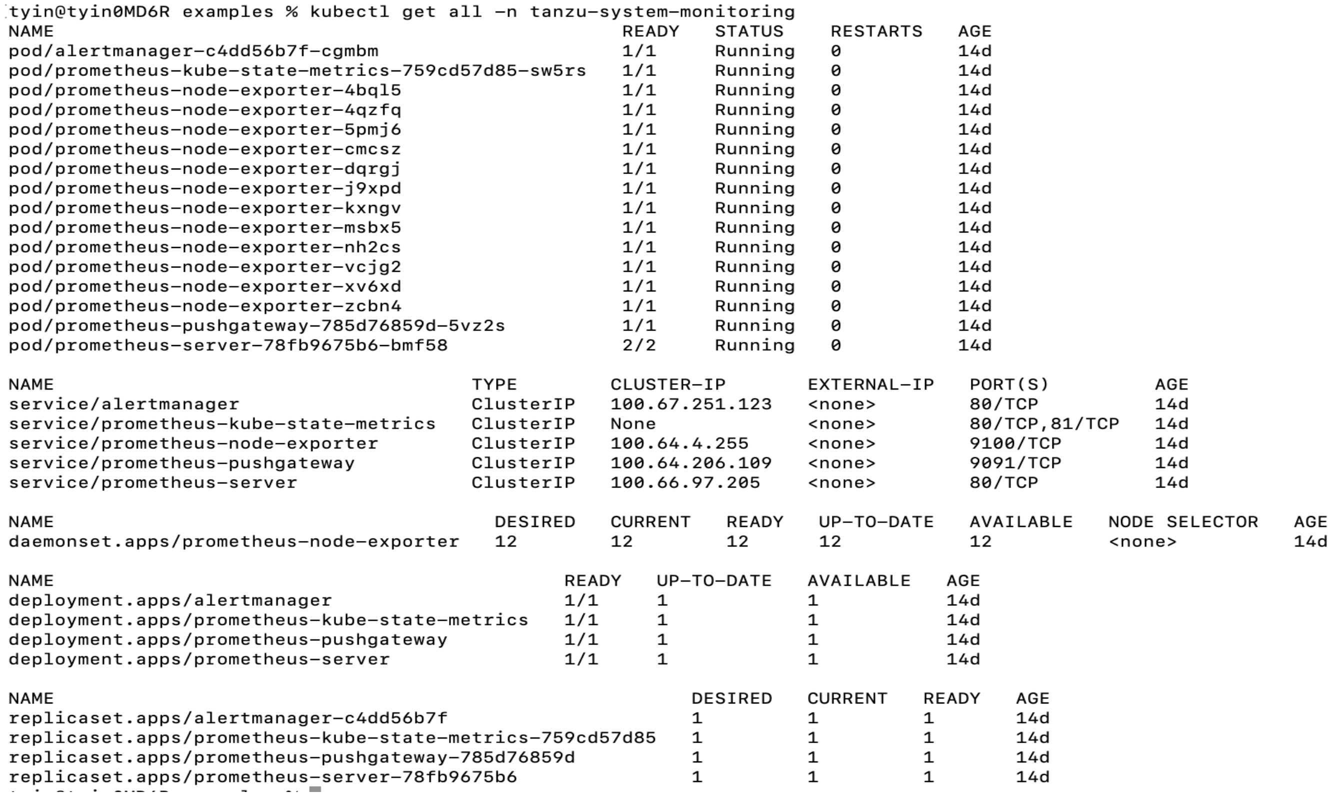
Prerequisites:

- Contour: see *Install Contour for Ingress Control*.
- cert-manager: see *Install cert-manager for Certificate Management*.

Deployment procedures:

1. Deploy Prometheus on workload cluster: we followed this *guide* to deploy Prometheus on your Tanzu Kubernetes Grid workload cluster.

Deploy Prometheus with the default configurations because it is on standalone management.



```
tyin@tyin0MD6R examples % kubectl get all -n tanzu-system-monitoring
NAME                                                     READY   STATUS    RESTARTS   AGE
pod/alertmanager-c4dd56b7f-cgmbm                         1/1     Running   0          14d
pod/prometheus-kube-state-metrics-759cd57d85-sw5rs       1/1     Running   0          14d
pod/prometheus-node-exporter-4bql5                       1/1     Running   0          14d
pod/prometheus-node-exporter-4qzfq                       1/1     Running   0          14d
pod/prometheus-node-exporter-5pmj6                       1/1     Running   0          14d
pod/prometheus-node-exporter-cmcsz                       1/1     Running   0          14d
pod/prometheus-node-exporter-dqrgj                       1/1     Running   0          14d
pod/prometheus-node-exporter-j9xpd                       1/1     Running   0          14d
pod/prometheus-node-exporter-kxngv                       1/1     Running   0          14d
pod/prometheus-node-exporter-msbx5                       1/1     Running   0          14d
pod/prometheus-node-exporter-nh2cs                       1/1     Running   0          14d
pod/prometheus-node-exporter-vcjg2                       1/1     Running   0          14d
pod/prometheus-node-exporter-xv6xd                       1/1     Running   0          14d
pod/prometheus-node-exporter-zcbn4                       1/1     Running   0          14d
pod/prometheus-pushgateway-785d76859d-5vz2s              1/1     Running   0          14d
pod/prometheus-server-78fb9675b6-bmf58                   2/2     Running   0          14d

NAME                                        TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)         AGE
service/alertmanager                        ClusterIP   100.67.251.123  <none>        80/TCP          14d
service/prometheus-kube-state-metrics       ClusterIP   None            <none>        80/TCP,81/TCP   14d
service/prometheus-node-exporter            ClusterIP   100.64.4.255    <none>        9100/TCP        14d
service/prometheus-pushgateway              ClusterIP   100.64.206.109  <none>        9091/TCP        14d
service/prometheus-server                   ClusterIP   100.66.97.205   <none>        80/TCP          14d

NAME                                        DESIRED   CURRENT   READY   UP-TO-DATE   AVAILABLE   NODE SELECTOR   AGE
daemonset.apps/prometheus-node-exporter    12        12        12      12           12          <none>          14d

NAME                                             READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/alertmanager                     1/1     1            1           14d
deployment.apps/prometheus-kube-state-metrics    1/1     1            1           14d
deployment.apps/prometheus-pushgateway           1/1     1            1           14d
deployment.apps/prometheus-server                1/1     1            1           14d

NAME                                                        DESIRED   CURRENT   READY   AGE
replicaset.apps/alertmanager-c4dd56b7f                      1         1         1       14d
replicaset.apps/prometheus-kube-state-metrics-759cd57d85    1         1         1       14d
replicaset.apps/prometheus-pushgateway-785d76859d           1         1         1       14d
replicaset.apps/prometheus-server-78fb9675b6                1         1         1       14d
```

Figure 3. tanzu-system-monitoring Namespace

2. Deploy Grafana on the workload cluster: follow the *instruction* to deploy Grafana on workload clusters. And then, verify whether Prometheus and Grafana are installed.

```
tanzu package installed list -A

NAMESPACE           NAME                                  PACKAGE-NAME                              PACKAGE-
      VERSION               STATUS
contour-package     contour                               contour.tanzu.vmware.com
      1.22.3+vmware.1-tkg.1          Reconcile succeeded
grafana-package     grafana                               grafana.tanzu.vmware.com
      7.5.16+vmware.1-tkg.1          Reconcile succeeded
prometheus-package prometheus                             prometheus.tanzu.vmware.com
      37.0+vmware.1-tkg.1          Reconcile succeeded
```

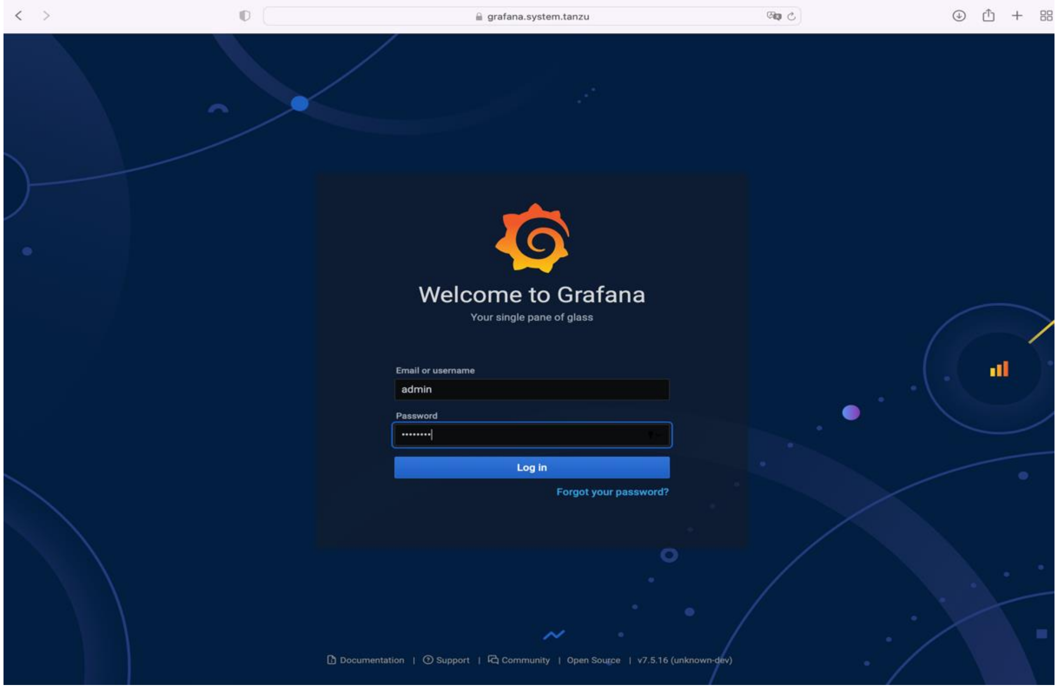
Figure 4. Prometheus and Grafana Package Installed Screenshot

The Grafana package creates a Contour HTTP Proxy object with a Fully Qualified Domain Name (FQDN) of grafana.system.tanzu.

Notes: Create an entry in your local /etc/hosts file that points an IP address of a worker node to this FQDN.

To access the Grafana dashboard, enter the url: *https://grafana.system.tanzu*

3. Use Grafana for monitoring.



grafana.system.tanzu

Welcome to Grafana

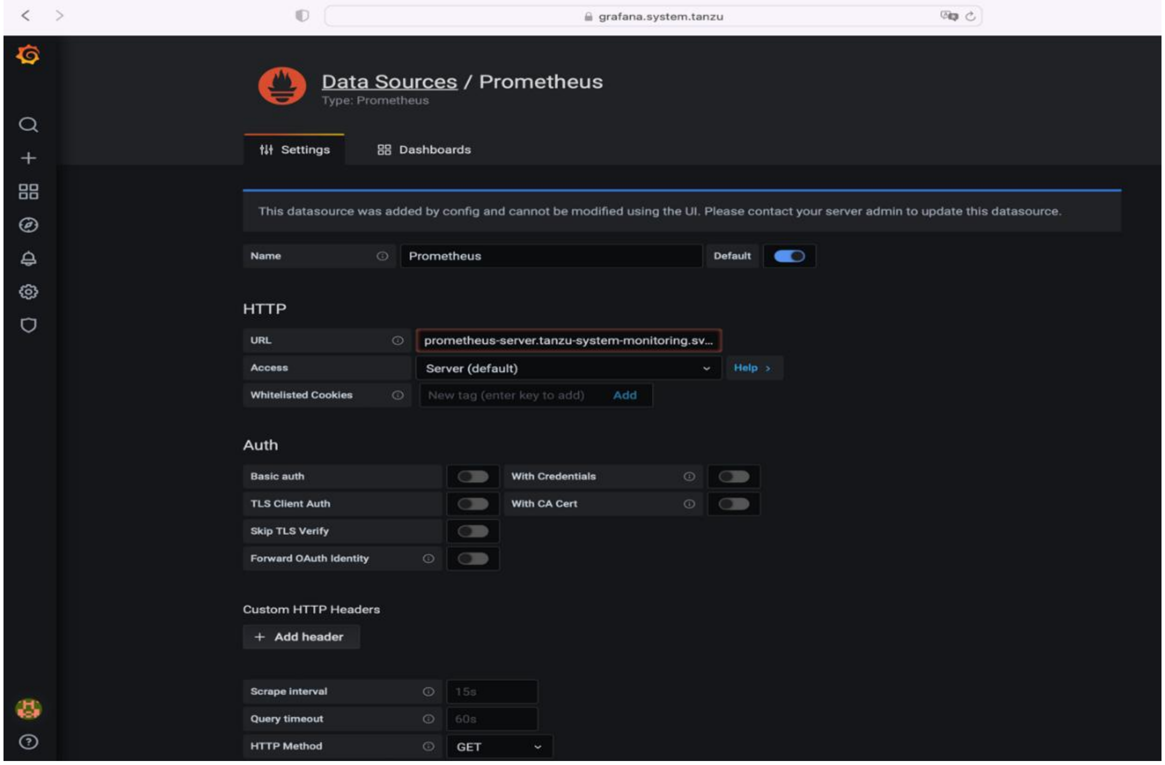
Your single pane of glass

Email or username: admin

Password: ••••••••

Figure 5. Grafana Login

Use the default user admin/admin to log in and then verify the Prometheus data sources.



grafana.system.tanzu

Data Sources / Prometheus

Type: Prometheus

This datasource was added by config and cannot be modified using the UI. Please contact your server admin to update this datasource.

Name: Prometheus

Default: on

HTTP

URL: prometheus-server.tanzu-system-monitoring.sv...

Access: Server (default)

Whitelisted Cookies: New tag (enter key to add)

Auth

Basic auth: off

With Credentials: off

TLS Client Auth: off

With CA Cert: off

Skip TLS Verify: off

Forward OAuth Identity: off

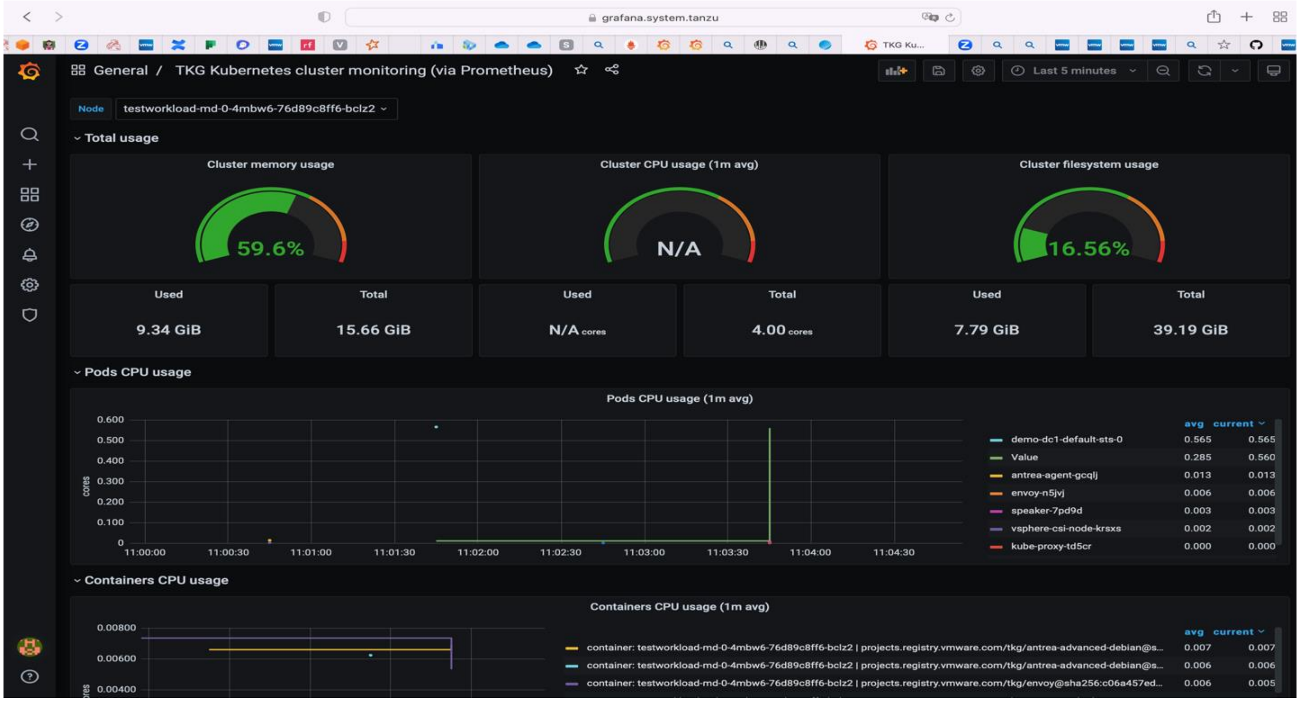
Custom HTTP Headers

Scrape interval: 15s

Query timeout: 60s

HTTP Method: GET

Figure 6. Prometheus DataSource



Used	Total	Used	Total	Used	Total
9.34 GiB	15.66 GiB	N/A cores	4.00 cores	7.79 GiB	39.19 GiB

	avg	current
demo-dc1-default-sts-0	0.565	0.565
Value	0.285	0.560
antrea-agent-gcqlj	0.013	0.013
envoy-n5jvj	0.006	0.006
speaker-7pd9d	0.003	0.003
vsphere-csi-node-krsxs	0.002	0.002
kube-proxy-td5cr	0.000	0.000

	avg	current
container: testworkload-md-0-4mbw6-76d89c8ff6-bclz2 | projects.registry.vmware.com/tkg/antrea-advanced-debian@s...	0.007	0.007
container: testworkload-md-0-4mbw6-76d89c8ff6-bclz2 | projects.registry.vmware.com/tkg/antrea-advanced-debian@s...	0.006	0.006
container: testworkload-md-0-4mbw6-76d89c8ff6-bclz2 | projects.registry.vmware.com/tkg/envoy@sha256:c06a457ed...	0.006	0.005

Figure 7. Tanzu Kubernetes Cluster Monitoring Dashboard

Solution Deployment and Validation

The Flink Kubernetes Operator and Cassandra Operator enable seamless deployment and management of Flink and Cassandra on Kubernetes respectively. By integrating them with Tanzu Kubernetes Grid, VMware's Kubernetes distribution, enterprises gain a fully validated and supported environment for running modern stream processing and storage workloads.

This solution validates the core capabilities of building and operating highly scalable real-time streaming analytics applications with strong SLAs on processing, availability, and supportability. The validation includes:

- Flink deployment: The Flink Kubernetes Operator simplifies submitting Flink applications as Kubernetes jobs by injecting them into the cluster as pods for automated scheduling, orchestration, and high availability.
- Flink job submission validation: The Flink Kubernetes Operator simplifies submission of Flink applications as Kubernetes jobs. It injects them into the cluster as pods for automated scheduling and orchestration.
- *Flink JobManager* high availability validation: The Flink JobManager, which oversees cluster resources and job scheduling, is made high available through Kubernetes deployments to ensure no single point of failure.
- Flink scalability with Reactive Mode: The Flink Operator supports reactive scaling to dynamically adjust cluster size based on workload metrics. It optimizes performance and costs through automated scaling.
- Cassandra deployment: The Cassandra Operator facilitates deployment of Cassandra clusters on Kubernetes, it also includes a suite of tools to ease and automate operational tasks.
- Stream application sample using Flink and Cassandra: An example application using Flink, Cassandra, Prometheus, and Grafana is included to demonstrate their integration in monitoring a stream processing pipeline.

Prometheus and Grafana are also deployed as part of the solution, providing metrics collection and visualization respectively, by bringing all components together on Tanzu Kubernetes Grid.

Flink Deployment

Install the certificate manager on your Kubernetes cluster to enable adding the webhook component (only needed once per Kubernetes cluster) to deploy the Flink Operator:

```
kubectl create -f https://github.com/jetstack/cert-manager/releases/download/v1.8.2/cert-manager.yaml

helm repo add flink-operator-repo https://downloads.apache.org/flink/flink-kubernetes-operator-1.3.1/
helm install flink-kubernetes-operator flink-operator-repo/flink-kubernetes-operator
Verify Flink Operator installed：
helm list
NAME                         NAMESPACE  REVISION  UPDATED                               STATUS        CHART
      APP VERSIO
flink-kubernetes-operator     default        1          2023-02-07 22:23:59.643829 +0800 CST  deployed
        flink-kubernetes-operator-1.3.1      1.3.1
```

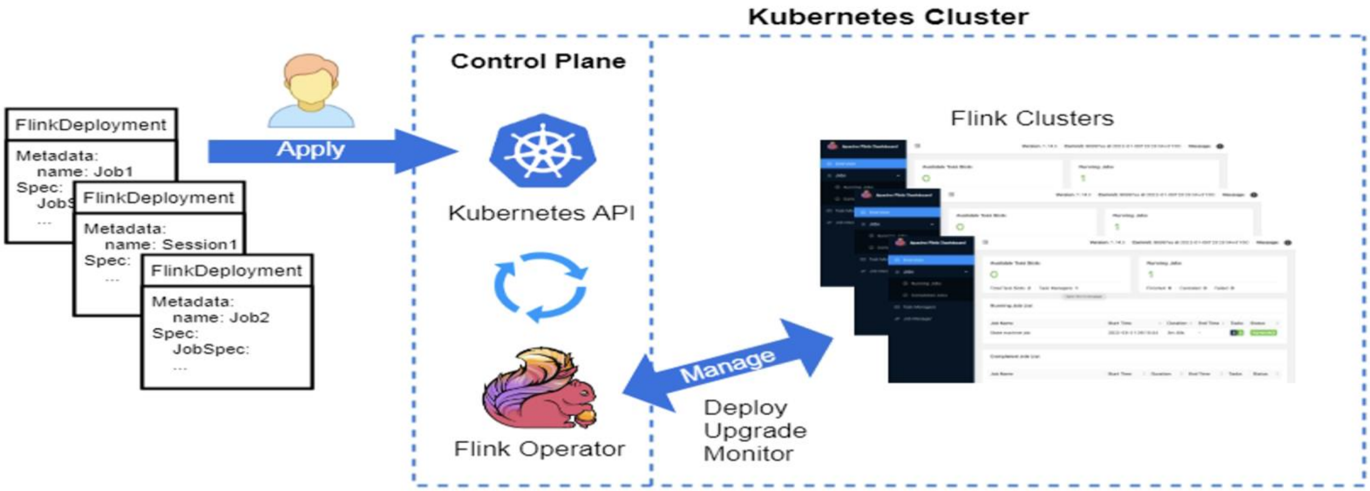


Figure 8. Flink Deployment

Flink Workload Validation

JobManager is the name of the central work coordination component of Flink. It has implementations for different resource providers, which differ on high-availability, resource allocation behavior, and supported job submission modes.

Job submission has two modes: Application Mode and Session Mode. We validated the job submissions of both modes respectively in the following chapter.

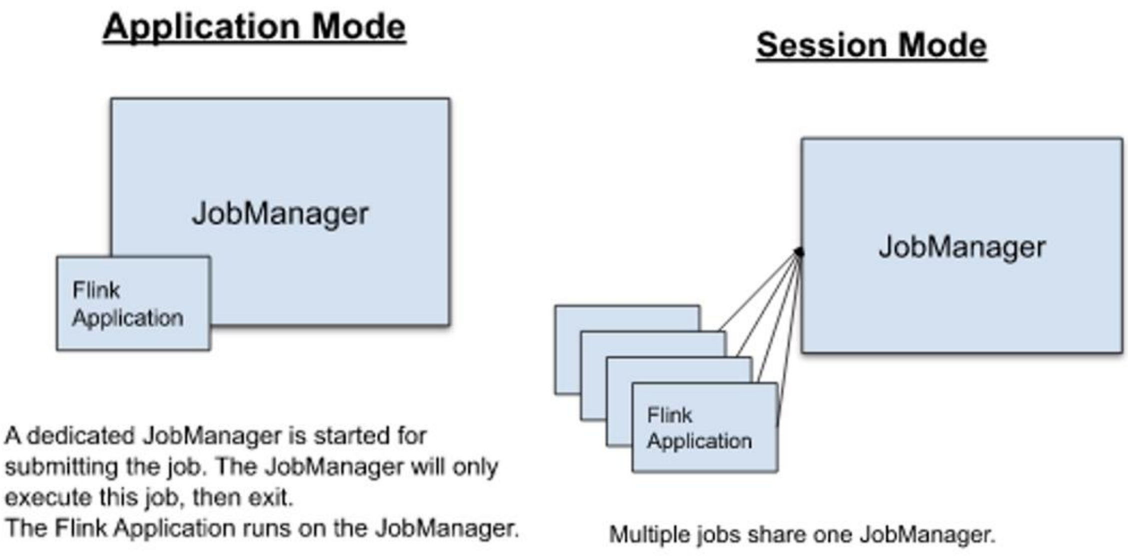


Figure 9. JobManager

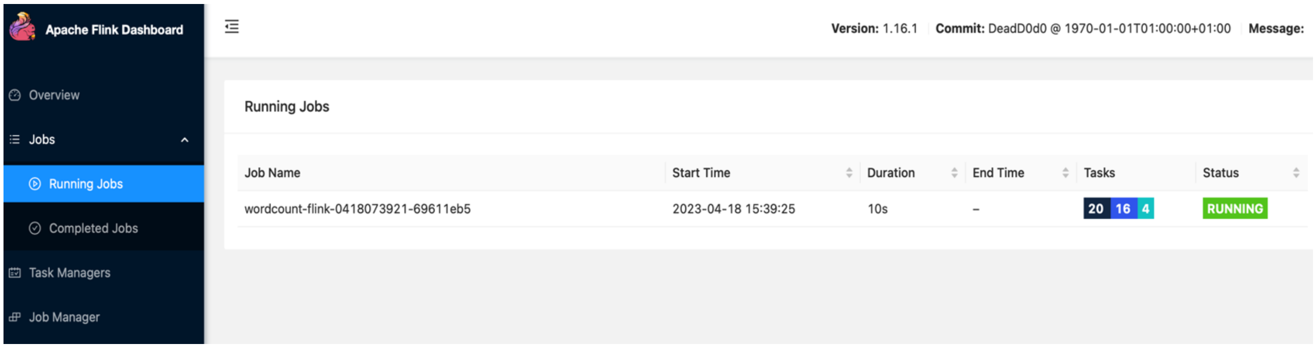
Application Mode Job

In Application Mode, Flink creates a cluster for each submitted application and runs the cluster exclusively for that application. The job's main method (or client) is run on the JobManager. Check out the *YAML* files.

The WordCount *Beam* application was first compiled with Flink runner and was packaged into a Flink application.

The Flink job can be viewed on the web UI through the beam-example-test service port.

```
tyin@tyin0MD6R flink-beam-example % kubectl port-forward service/beam-example-rest 8081:8081
Forwarding from 127.0.0.1:8081 -> 8081
Forwarding from [::1]:8081 -> 8081.
```



Version: 1.16.1 **Commit:** DeadD0d0 @ 1970-01-01T01:00:00+01:00 **Message:**

Running Jobs

Job Name	Start Time	Duration	End Time	Tasks	Status
wordcount-flink-0418073921-69611eb5	2023-04-18 15:39:25	10s	-	20 16 4	RUNNING

Figure 10. Beam Application in Application Mode

Session Mode Job

In Session mode, an existing cluster was used to run any submitted applications. One JobManager instance manages multiple jobs that share the same cluster of *TaskManagers*. This has the advantage of avoiding the resource overhead of spinning up a new cluster for each job. This is important in scenarios where jobs have short running time, as a high startup time would negatively impact the end-to-end user experience. For example, interactive analysis of short queries can benefit from Session Mode, as jobs can quickly perform computations using the existing resources.

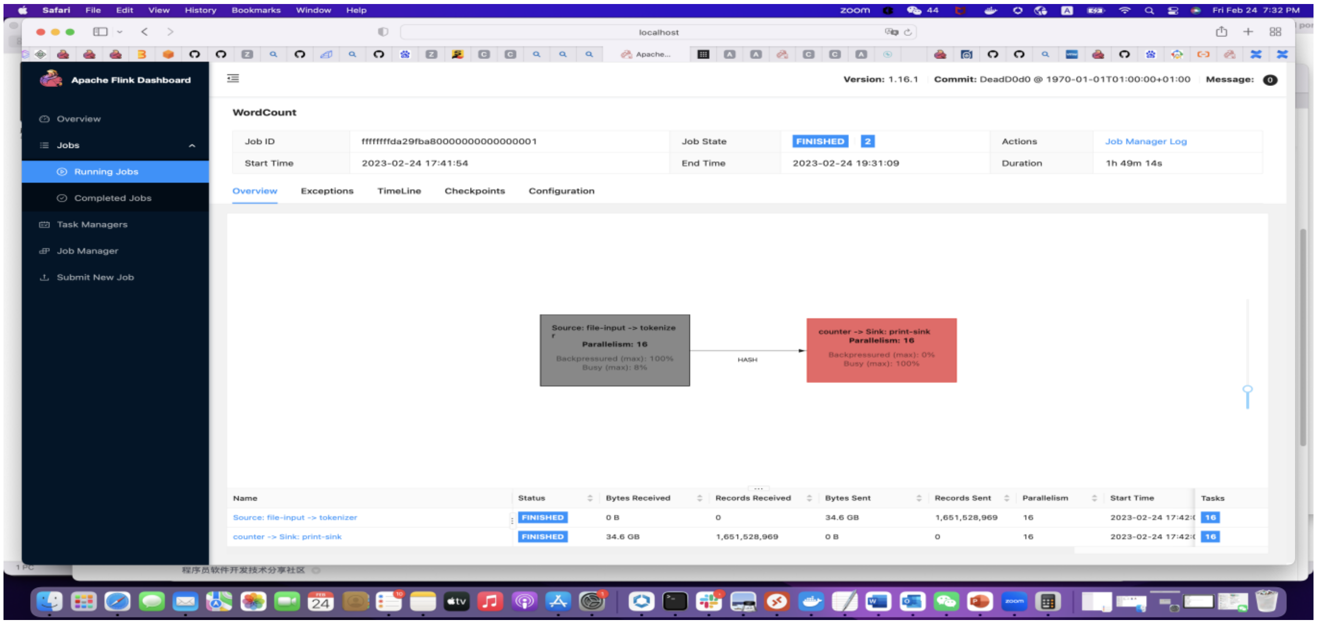
The limitation of Session Mode is that TaskManager slots are allocated by the *ResourceManager* on job submission and released once the job is completed. This means that there is competition for cluster resources between jobs. We first deployed a session cluster, check out the YAML file for detailed information.

```
Kubectl apply -f session-cluster.yaml
```

We started with the session manager configuration of 3 vCPU and 6GB memory. Then we tested with different parallelisms for the *WordCount* job, the input file is Wikipedia Dataset1 50GB with suffix 2. The file size was over 10GB. From the test results, the parallelism 32 configuration got the shortest duration.

Table 4. Session Mode Job Sample

Parallelism	TaskManager	Duration
8	1	3h6m52s
16	2	1h49m14s
32	4	1h8m33s



Version: 1.16.1 Commit: DeadD0d0 @ 1970-01-01T01:00:00+01:00 Message: 0

WordCount

Job ID	ffffffffda29fba80000000000000001	Job State	FINISHED 2	Actions	Job Manager Log
Start Time	2023-02-24 17:41:54	End Time	2023-02-24 19:31:09	Duration	1h 49m 14s

Name	Status	Bytes Received	Records Received	Bytes Sent	Records Sent	Parallelism	Start Time	Tasks
Source: file-input -> tokenizer	FINISHED	0 B	0	34.6 GB	1,651,528,969	16	2023-02-24 17:42:(	16
counter -> Sink: print-sink	FINISHED	34.6 GB	1,651,528,969	0 B	0	16	2023-02-24 17:42:(	16

Figure 11. WordCount Test

Flink JobManager High Availability

Apache Flink JobManager High Availability (HA) ensures that Flink clusters continue running the submitted jobs even in the event of a JobManager failure. Flink provides two HA service implementations:

- ZooKeeper: ZooKeeper HA services can be used with each Flink cluster deployment. They require a running ZooKeeper quorum.
- Kubernetes: Kubernetes HA services only work when running on Kubernetes.

HA Configuration

Check out the test scripts *here*.

Flink Kubernetes HA allows Flink clusters to be deployed to Kubernetes and to continue operating even in the event of a JobManager failure. To recover the submitted jobs, Flink persists metadata and the job artifacts. The HA data will be kept until the respective job either succeeds, is cancelled, or fails terminally. Once this happens, all the HA data, including the metadata stored in the HA services, will be deleted.

In the YAML file, the high-availability storageDir property must be set to the directory where the HA state will be stored.

With vSAN file service enabled, we can create ReadWriteMany persistent storage volumes dynamically.

```
---
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: flink-example-statemachine
spec:
  accessModes:
    - ReadWriteMany
  resources:
    requests:
      storage: 1Gi
```

We configured the below keys:

```
high-availability: org.apache.flink.kubernetes.highavailability.KubernetesHaServicesFactory
high-availability.storageDir: file:///opt/flink/volume/flink-ha
state.checkpoints.dir: file:///opt/flink/volume/flink-cp
state.savepoints.dir: file:///opt/flink/volume/flink-sp
```

Application HA Test

The Flink Kubernetes application HA test starts by setting variables, such as CLUSTER_ID, APPLICATION_YAML, and TIMEOUT. It then applies the YAML file specified in APPLICATION_YAML to create a Kubernetes deployment. The script then waits for the JobManager to start running and submits a job. After the job is running, the script waits for the logs to show that a checkpoint has been completed. The script then kills the JobManager and waits for the new JobManager to recover from the latest successful checkpoint. Finally, the script checks the operator log for errors and prints out a message indicating that the test is successful.

We also monitored the session-cluster-1 pod has restarted during the JobManager failure testing.

```
tyin@tyin0MD6R ~ % kubectl get pod
NAME                                             READY   STATUS    RESTARTS      AGE
pod/flink-kubernetes-operator-76dfd7fcf5-8xbkw   2/2     Running   0             64d
pod/session-cluster-1-7465757b49-pxrpb           1/1     Running   1 (24s ago)  117s
pod/session-cluster-1-taskmanager-1-1            1/1     Running   0             73s
pod/session-cluster-1-taskmanager-1-2            1/1     Running   0             73s
pod/session-deployment-8698f96848-xcc6k          1/1     Running   0             47d
```

Session Job HA Test

Similarly, the Flink Session cluster high availability is tested by killing the JobManager pod and ensuring that the cluster recovers successfully. The script does the following:

- Apply the YAML file for a Flink cluster and job, retrying up to 5 times.
- Wait up to TIMEOUT for the cluster status and job status to become READY and RUNNING.
- Get the job ID from the JobManager logs.
- Kill the JobManager pod.
- Wait for recovering logs and status until another checkpoint is completed.
- Check the operator log for errors.
- Print out a message indicating that the test was successful.

Flink Scalability with Reactive Mode

Flink offers reactive scaling, which automatically scales the size of a cluster up or down based on metrics. This automates and optimizes the provisioning and release of cluster resources in line with application workload demands. It adapts capacity seamlessly based on metrics to maximize performance, minimize costs, and ensure SLO/SLA compliance. This simplifies the management of stream processing at scale.

The Reactive Mode allows Flink users to implement a powerful autoscaling mechanism by having an external service monitor metrics such as aggregate CPU utilization, throughput, or latency. The Flink JobManager interacts with this external service. As soon as metrics surpass or fall below thresholds, additional TaskManagers can be added or removed from the Flink cluster.

Flink manages job parallelism that always maximizes values within constraints. Reactive scaling adjusts resources in line with workload demands, avoiding over or under provisioning. It keeps cluster size optimized for performance and cost, scaling out during load spikes and scaling in during lulls.

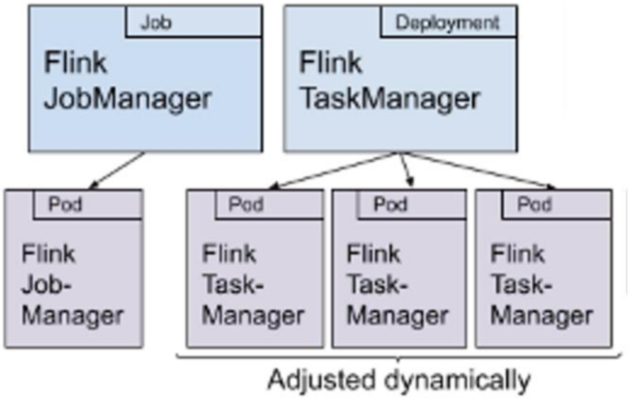


Figure 12. Flink Scalability with Reactive Mode

We used *basic-reactive.yaml* to verify scalability.

Deployment of Flink clusters on Kubernetes is only supported as a standalone application deployment. To enable elastic scaling, set the mode to standalone in the YAML file and enable the reactive scaling mode.

```
flinkConfiguration:

scheduler-mode: REACTIVE

mode: standalone

kubectl apply -f basic-reactive.yaml to validate the scalability function.
```

Manual Scaling

Scale the Tanzu Kubernetes Grid workload clusters and standalone management clusters using the following methods:

- Scale horizontally: For workload or standalone management clusters, you can manually scale the number of control plane and worker nodes. See *Scale a Cluster Horizontally*.
- Scale vertically: For workload clusters, you can manually change the size of the control plane and worker nodes. See *Scale a Cluster Vertically*.

In our validation, the workload cluster "testworkload" initially had 3 worker nodes. We can use Tanzu command line as below to scale the cluster with 6 worker nodes.

```
tyin@tyin0MD6R examples % tanzu cluster scale  testworkload -w 6
Workload cluster 'testworkload' is being scaled
```

And we use the command below to scale the task-manager to 3 replicas.

```
tyin@tyin0MD6R examples % kubectl scale

-replicas=3 deployments/basic-reactive-example-taskmanager

deployment.apps/basic-reactive-example-taskmanager scaled
```

We can see that TaskManager is scaling to 3 pods.

Automatic Scaling

Cluster Autoscaler can automatically scale the number of worker nodes in workload clusters deployed by a standalone management cluster. For more information, see *Scale Worker Nodes with Cluster Autoscaler*.

We can use the following command to scale TaskManager automatically based on CPU utilization metric.

```
kubectl autoscale deployment basic-reactive-example-taskmanager --min=1 --max=15 --cpu-percent=20
horizontalpodautoscaler.autoscaling/basic-reactive-example-taskmanager autoscaled

And
Kubectl get all
 …
NAME                                                                  REFERENCE
TARGETS    MINPODS    MAXPODS    REPLICAS    AGE
horizontalpodautoscaler.autoscaling/basic-reactive-example-taskmanager   Deployment/basic-reactive-example-
taskmanager    4%/20%    1          15         3           2m21s
```

K8ssandra Deployment

Deploying K8ssandra-Operator

We performed the following steps to deploy K8ssandra-operator on a single workload cluster:

1. Install k8sssandra-operator:

```
        helm install k8ssandra-operator k8ssandra/k8ssandra-operator -n k8ssandra-operator -- create-namespace

        helm list -n k8ssandra-operator
       NAME                   NAMESPACE              REVISION   UPDATED                                 STATUS

       k8ssandra-operator     k8ssandra-operator     1          2023-02-28 10:52:22.13637 +0800 CST deployed
       CHART          k8ssandra-operator-1.5.2
       APP VERSION    1.5.2
```

2. Check that there are two deployments in the k8ssandra-operator namespace:

```
kubectl -n k8ssandra-operator get deployment
NAME                 READY   UP-TO-DATE   AVAILABLE   AGE
cass-operator-controller-manager   1/1     1            1           2d
k8ssandra-operator                 1/1     1            1           2d
```

For more information, refer to *Install K8ssandra Operator*.

Cassandra Cluster

We chose Cassandra v4.0. 3 for the workload testing. Check out the test cluster YAML file *here.*
For detailed benchmark and configuration information, refer to running-k8ssandra-vmware-tanzu-kubernetes-grid-vmware-cloud-aws.

```
kubectl get svc -n k8ssandra-operator
NAME                                               TYPE           CLUSTER-IP       EXTERNAL-IP      PORT(S)
AGE
demo-dc1-additional-seed-service                   ClusterIP      None             <none>           <none>
43d
demo-dc1-all-pods-service                          ClusterIP      None             <none>
9042/TCP,8080/TCP,9103/TCP,9000/TCP                                                                        13d
demo-dc1-service                                   ClusterIP      None             <none>
9042/TCP,9142/TCP,8080/TCP,9103/TCP,9000/TCP                                                               13d
demo-dc1-stargate-service                          LoadBalancer   100.67.100.149    <none>
8080:30294/TCP,8081:31590/TCP,8082:31398/TCP,8084:30105/TCP,8085:30287/TCP,8090:31838/TCP,9042:30062/TCP   12d
demo-seed-service                                  ClusterIP      None             <none>           <none>
43d
k8ssandra-operator-cass-operator-webhook-service   ClusterIP      100.64.223.255   <none>           443/TCP
14d
k8ssandra-operator-webhook-service                 ClusterIP      100.64.241.64    <none>           443/TCP
14d
```

We changed the type of the demo-dc1-stargate-service service to LoadBalancer:

```
kubectl -n k8ssandra-operator patch service demo-dc1-stargate-service -p '{"spec": {"type":"LoadBalancer"}}'
service/demo-dc1-stargate-service patched
```

We used **100.67.100.149** for internal pod access and used **10.156.159.64** for Stargate service access.

```
kubectl get svc -n k8ssandra-operator
NAME                                               TYPE           CLUSTER-IP       EXTERNAL-IP      PORT(S)
AGE
demo-dc1-additional-seed-service                   ClusterIP      None             <none>           <none>
23d
demo-dc1-all-pods-service                          ClusterIP      None             <none>
9042/TCP,8080/TCP,9103/TCP,9000/TCP                                                                        23d
demo-dc1-service                                   ClusterIP      None             <none>
9042/TCP,9142/TCP,8080/TCP,9103/TCP,9000/TCP                                                               23d
demo-dc1-stargate-service                          LoadBalancer   100.67.100.149   10.156.159.64
8080:30828/TCP,8081:31581/TCP,8082:31336/TCP,8084:30232/TCP,8085:31656/TCP,8090:32022/TCP,9042:30204/TCP   22d
demo-seed-service                                  ClusterIP      None             <none>           <none>
23d
k8ssandra-operator-cass-operator-webhook-service   ClusterIP      100.64.223.255   <none>           443/TCP
24d
k8ssandra-operator-webhook-service                 ClusterIP      100.64.241.64    <none>           443/TCP
```

K8ssandra enables authentication and authorization by default. When the authentication is enabled, K8ssandra configures a new and default superuser. The username defaults to {metadata.name}-superuser. The credentials for the superuser are stored in a secret named {metadata.name}-superuser.

To retrieve the Cassandra cluster credential for future connections, use the following command:

```
kubectl get secret demo-superuser -o jsonpath="{.data.username}" -n k8ssandra-operator | base64 --decode
```

In our example, we deployed a cluster with name demo (replace the demo with the name configured for your running cluster).

To extract and decode the username secret, use the following command:

```
kubectl get secret demo-superuser -o jsonpath="{.data.password}" -n k8ssandra-operator | base64 –decode
```

Stargate has no specific credentials. It uses the same superuser as defined for Cassandra.

Stream Application Sample Running on Flink and Cassandra

We validated an example of the streaming pipeline. The workflow is as follows:

Figure 13. Stream Application Sample Running on Flink and Cassandra

The *TopSpeedWindowing* application is an example implementation of a streaming data processing application that uses Flink's windowing functionality to find the top speed of cars over a sliding time window, the code has been modified to persist the data to a Cassandra keyspace table.

Overall, this example provides a demonstration of how Flink's windowing functionality can be used to process the streaming data in a flexible and scalable way, it allows developers to perform complex computations on time-based data with ease.

Check out the *example code and YAML files*.

Schema of Destination Cassandra Keyspace Table

The output data stream topSpeeds contains tuples of the form (carId, speed, distance, timestamp) representing the highest speed recorded for each car during the window.

```
create keyspace example WITH replication = {'class': 'SimpleStrategy', 'replication_factor': '3'} AND
durable_writes = true;
CREATE TABLE example.topSpeeds (carid int, speed int, distance double, time bigint, PRIMARY KEY (carid,time))
WITH CLUSTERING ORDER BY (time DESC);

describe table example.topspeeds;

CREATE TABLE example.topspeeds (
    carid int PRIMARY KEY,
    distance double,
    speed int,
    time bigint
) WITH additional_write_policy = '99p'
    AND bloom_filter_fp_chance = 0.01
    AND caching = {'keys': 'ALL', 'rows_per_partition': 'NONE'}
    AND comment = ''
    AND compaction = {'class': 'org.apache.cassandra.db.compaction.SizeTieredCompactionStrategy',
'max_threshold': '32', 'min_threshold': '4'}
    AND compression = {'chunk_length_in_kb': '16', 'class': 'org.apache.cassandra.io.compress.LZ4Compressor'}
    AND crc_check_chance = 1.0
    AND default_time_to_live = 0
    AND gc_grace_seconds = 864000
    AND max_index_interval = 2048
    AND memtable_flush_period_in_ms = 0
    AND min_index_interval = 128
    AND read_repair = 'BLOCKING'
    AND speculative_retry = '99p';
```

Apache Flink Code to Aggregate and Persist Data in Cassandra Keyspace Table

Apache Cassandra Connector provides sinks that allow you to write data from a Flink data stream to a Cassandra database. The connector supports both batch and streaming data, and it can be used to write data to any Cassandra tables.

Perform the following steps to use the connector:

1. Create a CassandraSink object. This object takes a Flink data stream as an input and a Cassandra table as an output. You can then configure the sink with options such as the Cassandra host and port, the Cassandra keyspace, and the Cassandra table name.

```
final String username = "demo-superuser";
            final String password = "dJLCx7Y0opoIfEarDJTV";
            final String contactpoint = params.getContactPoint().orElse("127.0.0.1");
            try {
                CassandraSink.addSink(topSpeeds)
                        .setQuery(
                                "INSERT INTO example.topspeeds(carid, speed,distance, time) values (?, ?, ?, ?);")
                        .setClusterBuilder(
                                new ClusterBuilder() {
                                    private static final long serialVersionUID =
                                            2793938419775311824L;

                                    public Cluster buildCluster(Cluster.Builder builder) {
                                        return builder.addContactPoints(contactpoint)
                                                .withPort(9042)
                                                .withCredentials(username, password)
                                                .withoutJMXReporting()
                                                .build();
                                    }
                                })
                        .build();
            } catch (Exception e) {
                System.out.println("Error connecting to cluster" + e.getMessage());
            }
```

2. To use this connector, add the following dependency to the pom.xml in your project:

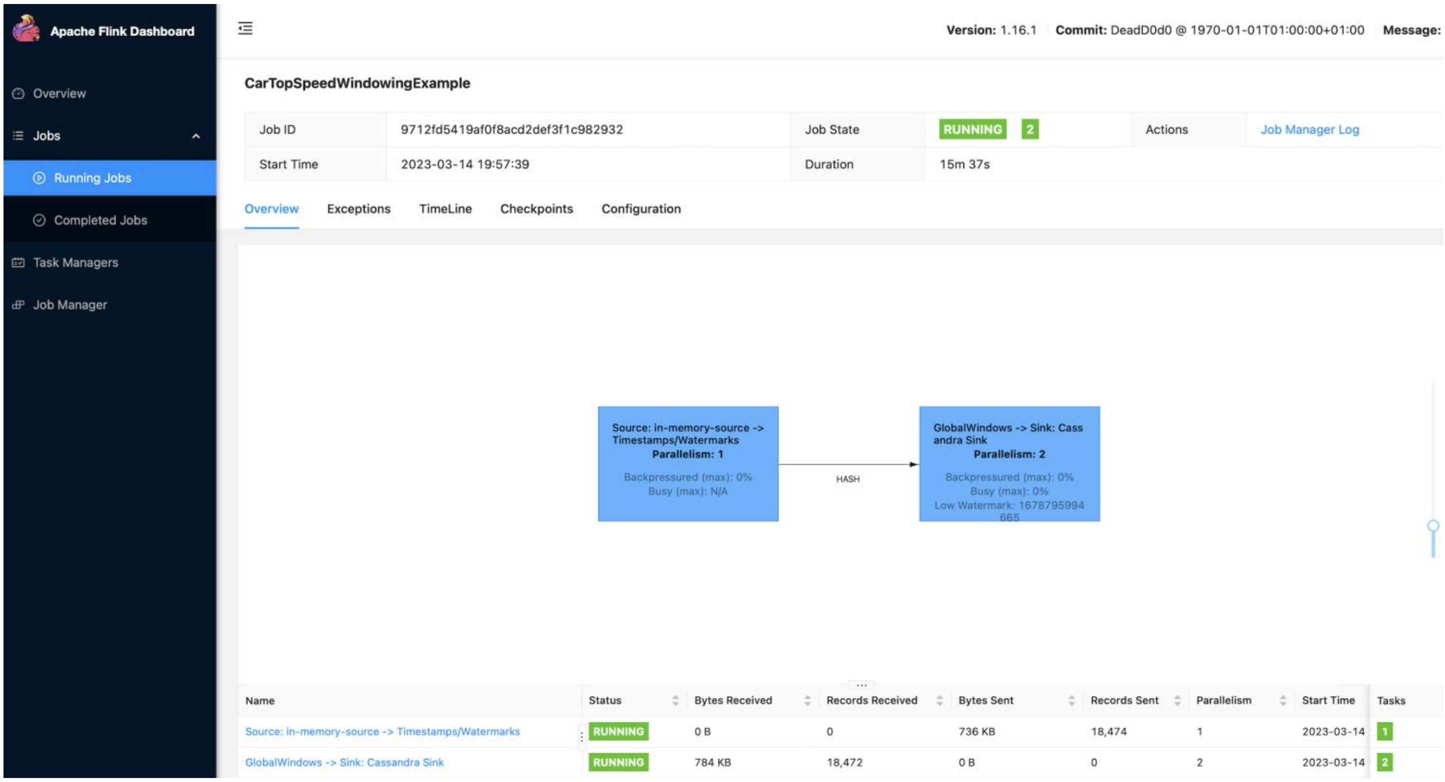
```
<dependency>
    <groupId>org.apache.flink</groupId>
    <artifactId>flink-connector-cassandra_2.12</artifactId>
    <version>1.16.1</version>
</dependency>
```

3. Build an executable jar. Since we are running in a containered environment, we need to Create Executable Fat Jar with Maven Shade Plugin.
4. Build an image that includes the jar.

Job Running

We used the following YAML file to submit the carTopSpeed job and sink to Cassandra.

```
Kubectl apply -f topcarwithcassparam.yaml
```



CarTopSpeedWindowingExample

Job ID	9712fd5419af0f8acd2def3f1c982932	Job State	RUNNING 2	Actions	Job Manager Log
Start Time	2023-03-14 19:57:39	Duration	15m 37s		

Name	Status	Bytes Received	Records Received	Bytes Sent	Records Sent	Parallelism	Start Time	Tasks
Source: in-memory-source -> Timestamps/Watermarks	RUNNING	0 B	0	736 KB	18,474	1	2023-03-14	1
GlobalWindows -> Sink: Cassandra Sink	RUNNING	784 KB	18,472	0 B	0	2	2023-03-14	2

Figure 14. Job Running Sample

Result Verification

In our example, we deployed a cluster with name demo (replace the demo with the name configured for your running cluster).

After the Flink job starts and persists results to database, we can use ‘cqlsh’ to connect to the Cassandra keyspace and then query and review the aggregated data as illustrated below:

```
tyin@tyin0MD6R bin % /cqlsh 10.156.159.164 -u demo-superuser --request-timeout=6000
Password:
Connected to demo at 10.156.159.164:9042.
[cqlsh 6.8.0 | Cassandra 4.0.4 | CQL spec 3.4.5 | Native protocol v4]
Use HELP for help.
demo-superuser@cqlsh> select count(*) from example.topspeeds ;

 count
-------
  3546
```

However, as the number of records inserted increases quickly, using ‘cqlsh’ will eventually generate a timeout error. In this case, we can use tablestats to check the number of records written to the topspeeds table.

```
tyin@tyin0MD6R examples % kubectl exec -it pod/demo-dc1-default-sts-0 -n k8ssandra-operator -c cassandra --
nodetool -u demo-superuser -pw dJLCx7Y0opoIfEarDJTV tablestats example.topspeeds
Total number of tables: 42
----------------
Keyspace: example
        Read Count: 0
        Read Latency: NaN ms
        Write Count: 35040986
        Write Latency: 0.009253590552503289 ms
        Pending Flushes: 0
                Table: topspeeds
                SSTable count: 3
                Old SSTable count: 0
                Space used (live): 150225664
```

```
Space used (total): 150225664
Space used by snapshots (total): 0
Off heap memory used (total): 118112
SSTable Compression Ratio: 0.6663820948690086
Number of partitions (estimate): 4000
Memtable cell count: 4552412
Memtable data size: 77782389
Memtable off heap memory used: 0
Memtable switch count: 6
Local read count: 0
Local read latency: NaN ms
Local write count: 35040986
Local write latency: NaN ms
Pending flushes: 0
Percent repaired: 0.0
Bytes repaired: 0.000KiB
Bytes unrepaired: 214.478MiB
Bytes pending repair: 0.000KiB
Bloom filter false positives: 0
Bloom filter false ratio: 0.00000
Bloom filter space used: 7536
Bloom filter off heap memory used: 7512
Index summary off heap memory used: 768
Compression metadata off heap memory used: 109832
Compacted partition minimum bytes: 11865
Compacted partition maximum bytes: 379022
Compacted partition mean bytes: 41255
Average live cells per slice (last five minutes): NaN
Maximum live cells per slice (last five minutes): 0
Average tombstones per slice (last five minutes): NaN
Maximum tombstones per slice (last five minutes): 0
Dropped Mutations: 0
Droppable tombstone ratio: 0.00000
```

.

Best Practices

- It is recommended to deploy multiple workload cluster for production for better resource isolation and monitoring.
- For workload cluster deployment, start from a small number of nodes to tune the parameters and then scale up gradually.
- For K8ssandra cluster deployment, the Stargate heap size should match the Cassandra pod, the proper size can be tuned through the throughput or latency workload testing.
- For the Flink cluster deployment, start from a small number of nodes and small worker node size to tune the parameters and then scale up gradually.

Conclusion

Flink and Cassandra are two popular open-source tools that work together appropriately for modern applications. By combining Flink and Cassandra, enterprises can build integrated real-time streaming analytics pipelines. The Flink Kubernetes Operator, Cassandra Operator, Prometheus, and Grafana integrate seamlessly on Tanzu Kubernetes Grid. This solution thus streamlines building and running real-time workloads on Kubernetes, which allows IT administrators to enable fast application deployment, achieve better scalability for performance, ensure high availability, governance and lower TCO expenditure.

Reference

For more information, you can explore the following resources:

- *VMware vSphere*
- *VMware vSAN*
- *VMware Tanzu Kubernetes Grid*
- *Apache Flink docs*
- *K8ssandra docs*

About the Author

Ting Yin, Senior Technical Marketing Architect in the Workload Technical Marketing Team of the Cloud Infrastructure Big Group, wrote the original version of this paper. The following reviewers also contributed to the paper contents:

- Chen Wei, Director of the Workload Technical Marketing Team in VMware
- Catherine Xu, Senior Manager of the Workload Technical Marketing Team in VMware

VMware, Inc. 3401 Hillview Avenue Palo Alto CA 94304 USA Tel 877-486-9273 Fax 650-427-5001 www.vmware.com.