Optimizing Trino on Kubernetes:

Helm Chart Enhancements for Resilience and Security

trino SUMMIT

December 11 - 12, 2024

Speakers:

Sebastian Daberdaku	Cardo AI
Jan Waś	Starburst

About the speakers

Sebastian Daberdaku

Data Engineering Tech Lead

CARDO AI

opportunity from complexity

Jan Waś

Software Engineer

Starburst

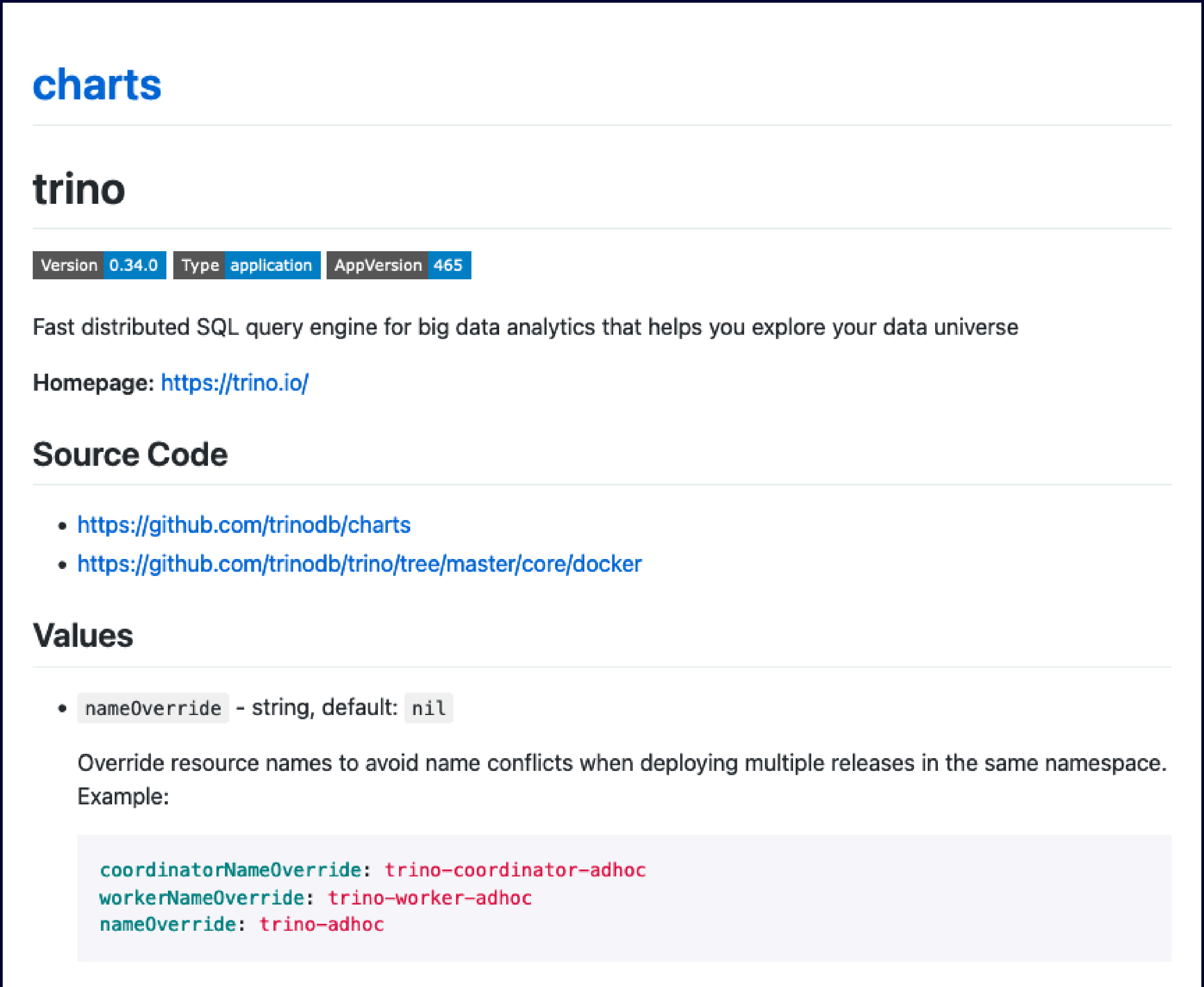
Trino Community Helm Chart

https://github.com/trinodb/charts.git

```
helm repo add trino https://trinodb.github.io/charts/
helm install my-trino trino/trino --version 0.34.0
```

Documentation

https://trinodb.github.io/charts/



charts

trino

Version 0.34.0 Type application AppVersion 465

Fast distributed SQL query engine for big data analytics that helps you explore your data universe

Homepage: https://trino.io/

Source Code

- https://github.com/trinodb/charts
- https://github.com/trinodb/trino/tree/master/core/docker

Values

- `nameOverride` - string, default: `nil`

  Override resource names to avoid name conflicts when deploying multiple releases in the same namespace. Example:

  ```
  coordinatorNameOverride: trino-coordinator-adhoc
  workerNameOverride: trino-worker-adhoc
  nameOverride: trino-adhoc
  ```

Development guidelines

1. Does it work correctly?
2. Is it robust?
3. Is it easy to use?

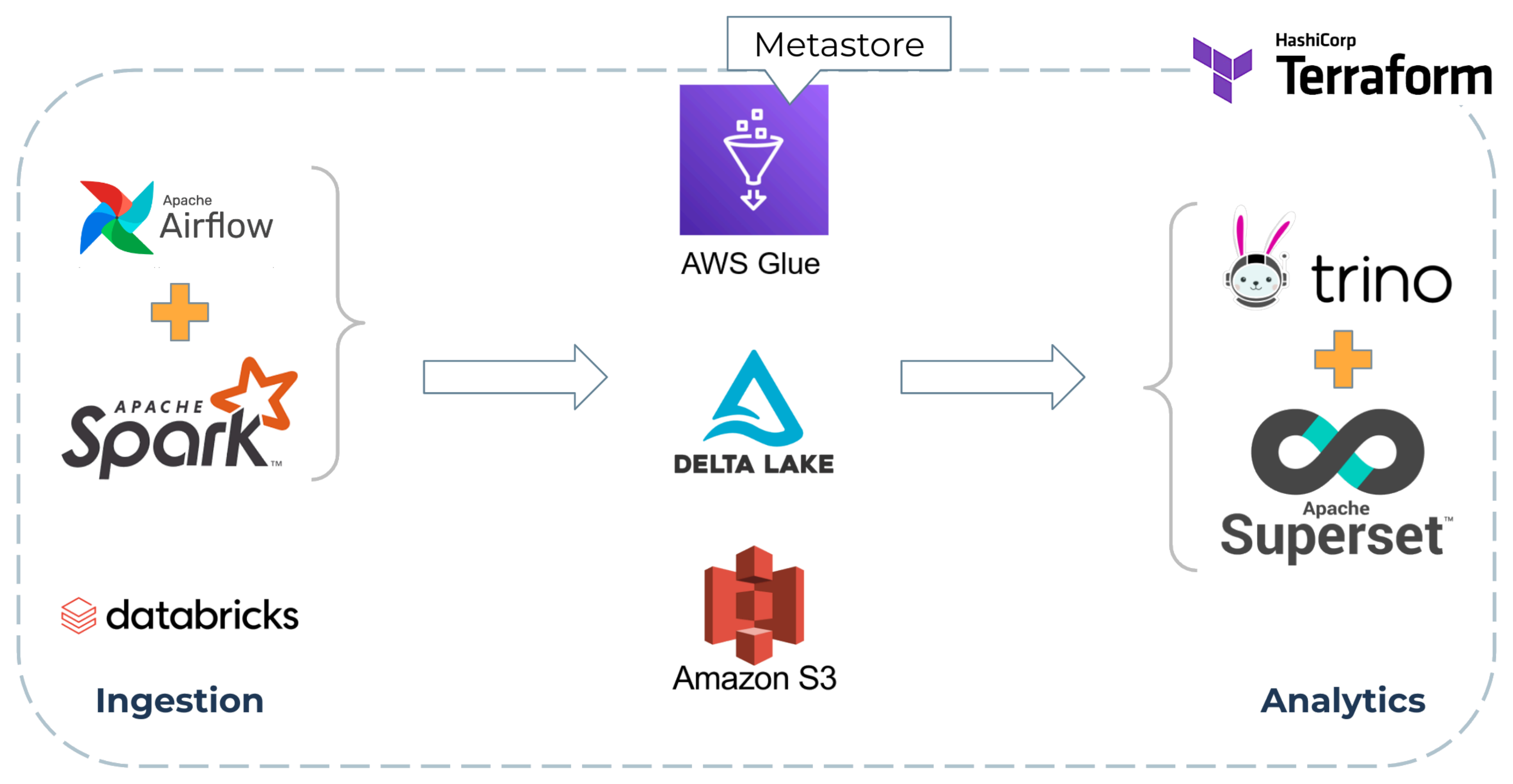
Test framework

1. Make changes with confidence
2. Test every Pull Request
3. Test locally
4. Test end to end

Test suites

1. Defaults
2. Single node
3. Complete values
4. Multiple releases in a single namespace
5. Access control
6. Exchange manager
7. Graceful shutdown
8. Resource groups

Lakehouse tech-stack @ Cardo AI



Trino usage @ Cardo AI

- We have been using Trino in production since 2022;
- Mostly short-lived, real-time queries;
- Initially deployed on AWS EKS with custom (forked) Helm Chart;
- Many of the presented features were developed in our fork of the Chart;
- Frequent Chart updates required to keep-up with the fast-paced Trino releases;
- Finally decided to "donate" these features to the official Chart.

Rendering default TPC-H and TPC-DS catalogs optional

- Initially, the TPC-H and TPC-DS catalogs were always created by the Helm Chart.
- These catalogs are useful for benchmarking a given Trino deployment configuration. After configuring the Trino cluster, users might want to drop these catalogs.
- Mounting static catalogs from a `configmap` conflicts with the Dynamic Catalogs feature which allows users to create catalogs at runtime with SQL statements which must be persisted on a mounted volume.
- Now, these catalogs can be easily disabled by Helm Chart users if they wish to do so.

Disabling TPC-H and TPC-DS catalogs

```yaml
# values.yaml
catalogs:
  tpch: |
    connector.name=tpch
    tpch.splits-per-node=4
  tpcds: |
    connector.name=tpcds
    ptcds.splits-per-node=4
```

Disabling TPC-H and TPC-DS catalogs

```yaml
# values.yaml
catalogs:
  tpch: null
  tpcds: null
```

Templating support for `additionalConfigFiles`

- The `additionalConfigFiles` properties on the coordinator and worker configurations allow users to add additional config files in the corresponding default configuration directories.
- These configurations are now templated:

```
# configmap-coordinator.yaml
apiVersion: v1
kind: ConfigMap
...
data:
...
{{- range $fileName, $fileContent := .Values.coordinator.additionalConfigFiles }}
  {{ $fileName }}: |
    {{- tpl $fileContent $ | nindent 4 }}
{{- end }}
```

File group provider example

- The group file must contain a list of groups and members, one per line, separated by a colon. Users are separated by a comma.

```yaml
# values.yaml
auth:
  refreshPeriod: 60s
  groups: |-
    admin:admin@example.com
    group1:user1@example.com,user2@example.com
```

File group provider example

```
# values.yaml
coordinator:
 additionalConfigFiles:
   group-provider.properties: |-
     group-provider.name=file
     file.group-file={{- .Values.server.config.path -}}/group-provider.db
     file.refresh-period={{- .Values.groupProvider.refreshPeriod }}
   group-provider.db: |-
     {{- range $k, $v := .Values.groupProvider.groups }}
     {{- printf "%s:%s\n" $k (join "," $v) }}
     {{- end }}
...
groupProvider:
  refreshPeriod: 60s
  groups:
    admin:
      - admin@example.com
    group1:
      - user1@example.com
      - user2@example.com
```

Resulting group-provider.db

```
# group-provider.db
admin:admin@example.com
group1:user1@example.com,user2@example.com
```

Worker graceful shutdown

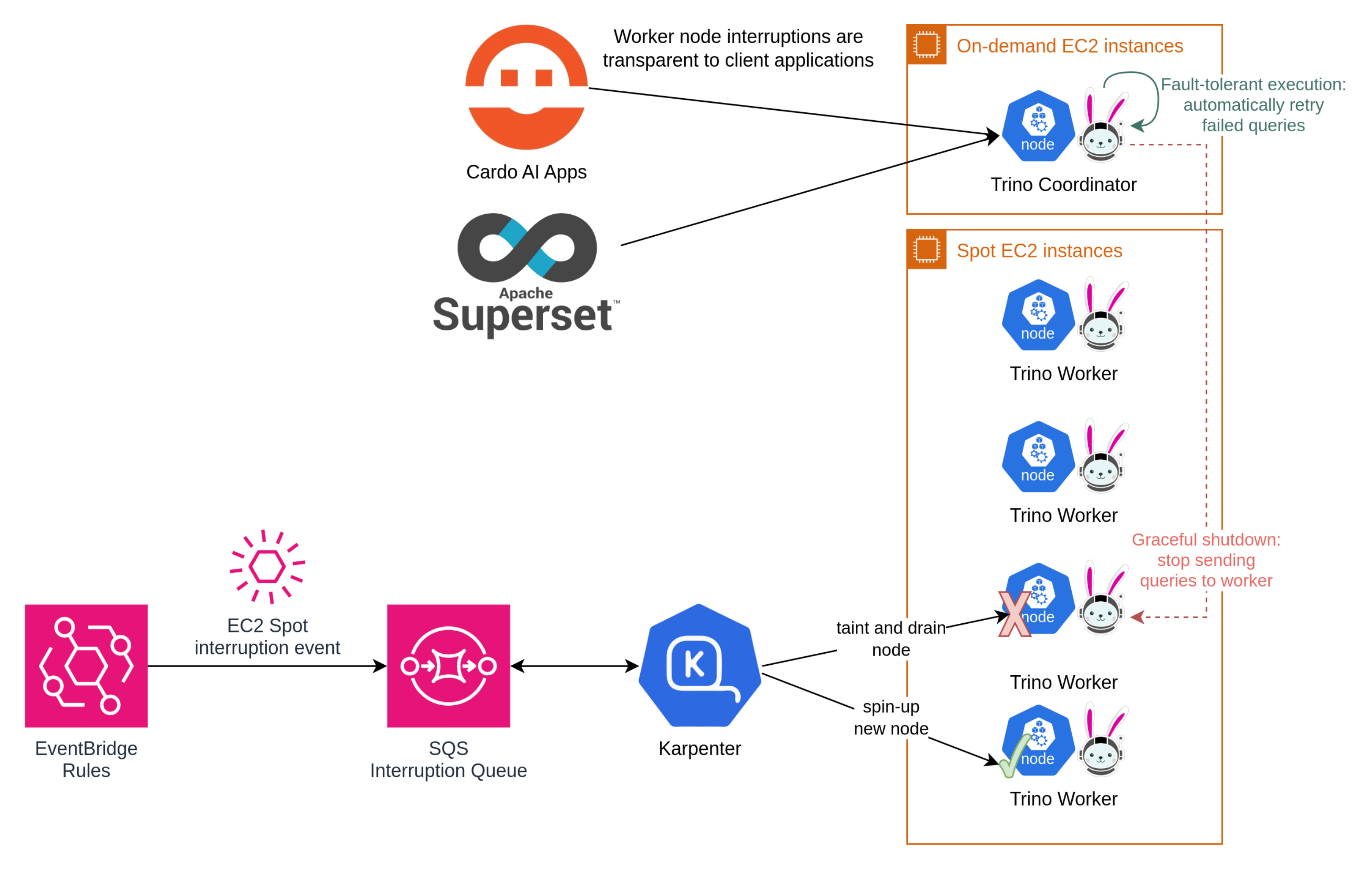
- Trino has a graceful shutdown API that can be used on workers in order to ensure that they terminate without affecting running queries, given a sufficient grace period.
- Once the API is called, the worker performs the following steps:
    1. Go into `SHUTTING_DOWN` state.
    2. Sleep for `shutdown.grace-period`. After this, the coordinator is aware of the shutdown and stops sending tasks to the worker.
    3. Block until all active tasks are complete.
    4. Sleep for the grace period again in order to ensure the coordinator sees all tasks are complete.
    5. Shutdown the application.

Enable worker graceful shutdown

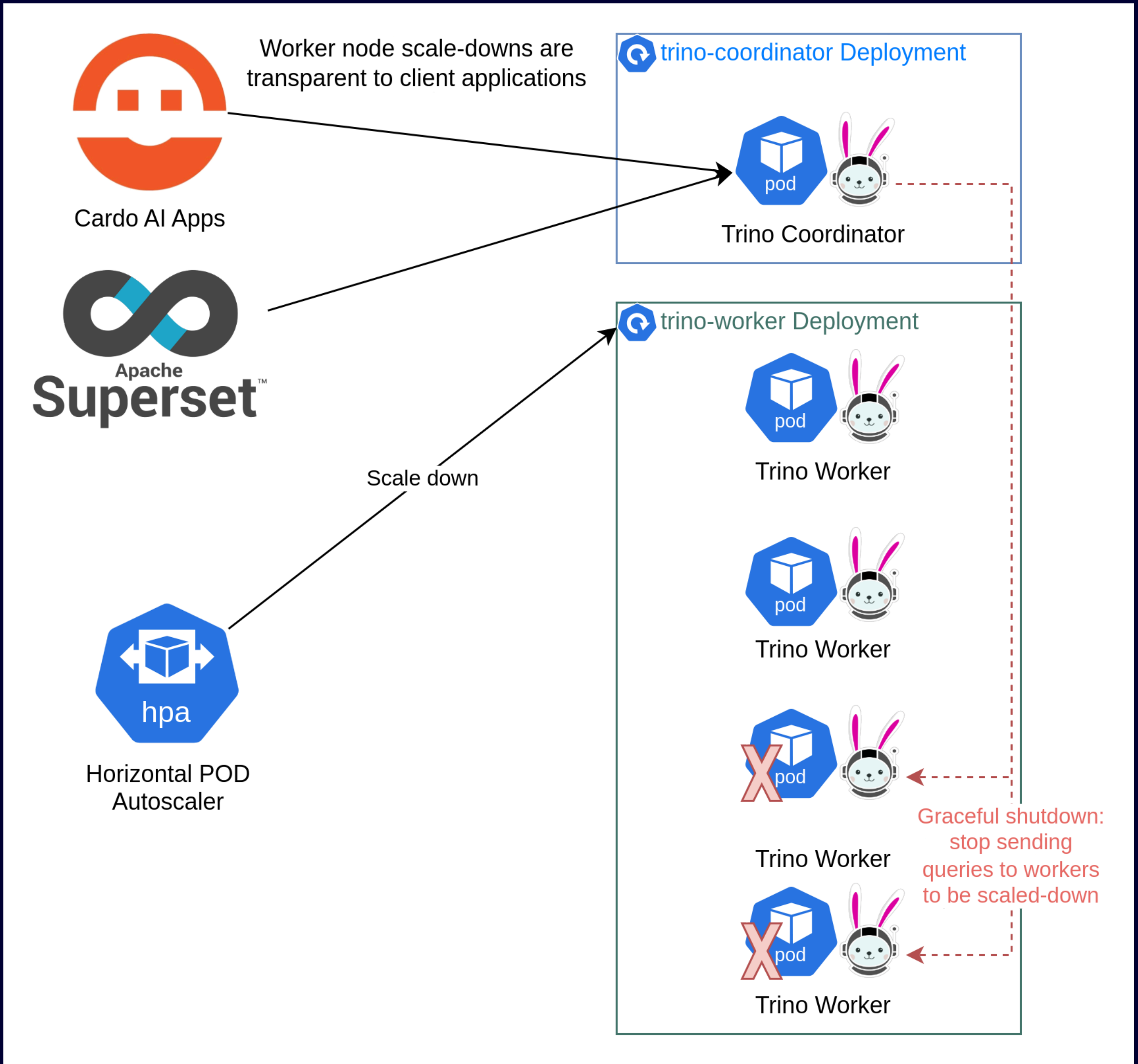
If enabled, the worker graceful shutdown configuration will:

- Add a preStop lifecycle event to all worker Pods;
- Configures the `shutdown.grace-period` property;
- Configure the workers' `accessControl` since the default system access control does not allow graceful shutdowns;
- Validate the `worker.terminationGracePeriodSeconds` value (which must be at least 2 × `shutdown.grace-period`);
- Ensure that `worker.lifecycle` is not set.

Cost-effective deployment with worker graceful shutdown



Autoscaling with worker graceful shutdown


Worker node scale-downs are transparent to client applications
Cardo AI Apps
Apache Superset™
hpa
Horizontal POD Autoscaler
Scale down
trino-coordinator Deployment
pod
Trino Coordinator
trino-worker Deployment
pod
Trino Worker
pod
Trino Worker
pod
Trino Worker
pod
Trino Worker
Graceful shutdown: stop sending queries to workers to be scaled-down

Testing worker graceful shutdown

To test the correctness of the feature the following test was created:

1. A **`kubectl`** container tails the worker Pod's logs and looks for the "Shutdown requested" message.
2. Another **`kubectl`** container deletes the worker Pod triggering the **`pre-stop`** lifecycle hook.

Testing worker graceful shutdown

```
# test-graceful-shutdown.yaml
apiVersion: v1
kind: Pod
...
  containers:
    - name: check-logs
      image: bitnami/kubectl:latest
      command: [ "sh", "-c" ]
      args:
        - >-
          WORKER_POD=$(cat /pods/worker-pod.txt) &&
          kubectl logs ${WORKER_POD}
          --follow
          --container=trino-worker
          --namespace={{ .Release.Namespace }}
          | grep --max-count=1 "Shutdown requested"
...
    - name: trigger-graceful-shutdown
      image: bitnami/kubectl:latest
      command: [ "sh", "-c" ]
      args:
        - >-
          sleep 5 &&
          WORKER_POD=$(cat /pods/worker-pod.txt) &&
          kubectl delete pod
          ${WORKER_POD}
          --namespace={{ .Release.Namespace }}
...
```

Enabling JMX Exporter on Trino Workers

Trino exposes a large number of different metrics via Java Management Extensions:

- JVM metrics (heap size, thread count)
- Trino cluster and node statistics
- Trino query metrics (number of active, queued, failed, etc.)
- Trino task metrics (input data bytes and rows)
- Connector metrics

Initially, the JMX Exporter could be enabled only on the Coordinator.

- JMX Exporter support was also added to Workers.

Testing JMX Exporter

- Install Prometheus in `kind` K8s cluster.
- Install Trino with JMX Exporter feature enabled for both coordinator and workers.
- Verify that JMX metrics are collected by Prometheus.

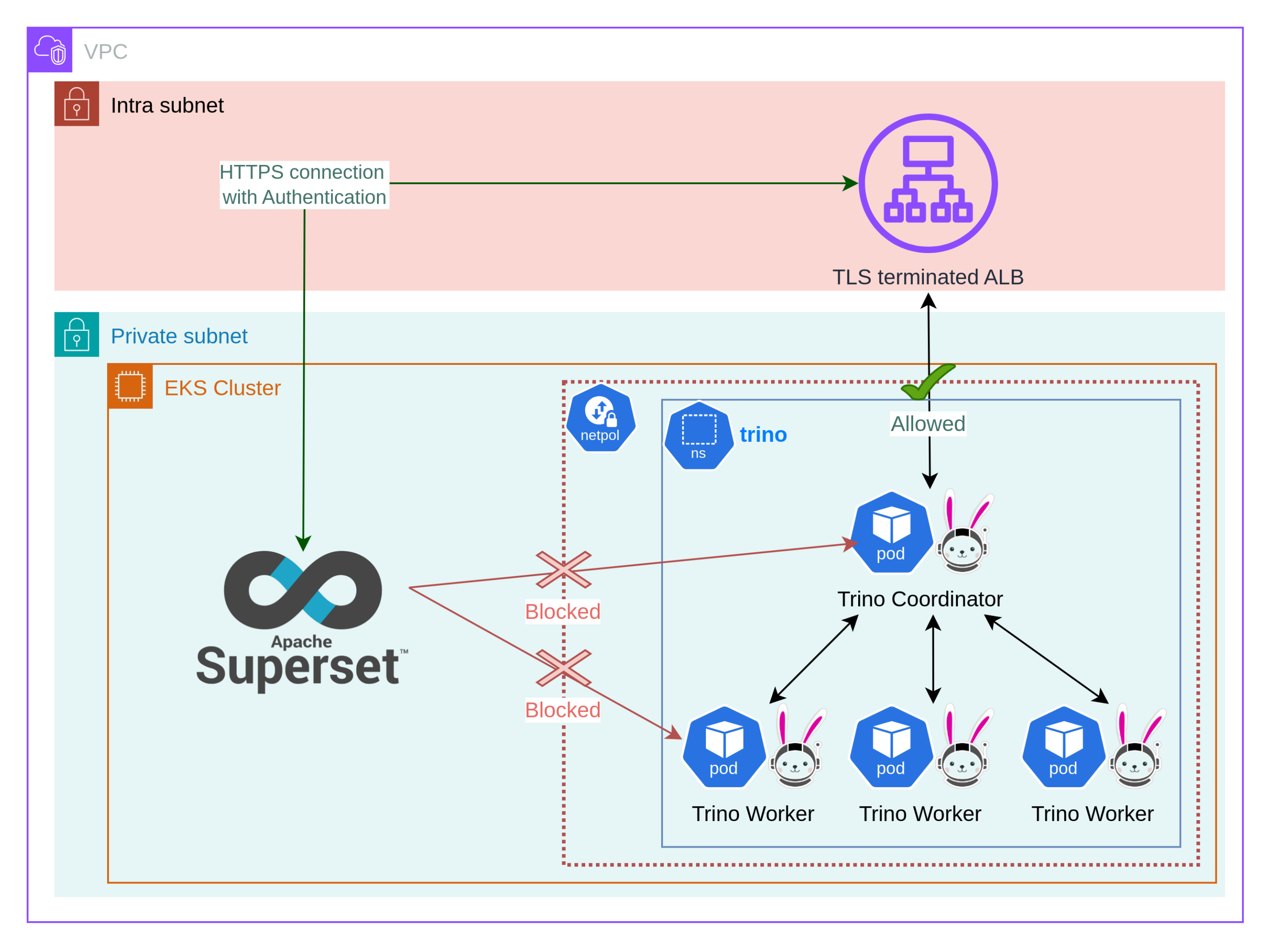
`NetworkPolicy` protection

- Trino supports multiple authentication types to ensure all users of the system are authenticated. Different authenticators allow user management in one or more systems.
- All authentication requires secure connections using TLS and HTTPS or process forwarding enabled.
- To configure Trino with TLS there are two alternatives:
  1. **Use a TLS-terminated Load Balancer or proxy (preferred)**;
  2. Secure Trino directly with valid a TLS certificate (must manage certificates, increased CPU usage).

`NetworkPolicy` protection

- To prevent unauthorized connections to Trino from within the Kubernetes cluster, a `NetworkPolicy` can be used.
- On EKS requires VPC CNI Plugin.
- Can be used to only allow ingress traffic to Trino from Pods with certain labels or from given CIDR blocks.

TLS-terminated ALB with NetworkPolicy protection



Testing `NetworkPolicy` protection

Connections from unauthorized Pods will time-out.

```yaml
# test-networkpolicy.yaml
apiVersion: v1
kind: Pod
...
  containers:
    - name: check-connection
      image: {{ include "trino.image" . }}
      command: [ "/bin/bash", "-c" ]
      args:
        - >-
          curl
          {{ include "trino.fullname" . }}.{{ .Release.Namespace }}:{{ .Values.service.port }}
          --head
          --fail
          --connect-timeout 10
          --max-time 10
          2>&1 | grep -q "timed out"
...
```

Future work:

Worker Autoscaling with KEDA

- The JMX Export, fault-tolerant query execution, and worker graceful shutdown can be used to implement advanced worker autoscaling.

```
apiVersion: keda.sh/v1alpha1
kind: ScaledObject
metadata:
  name: trino-worker
spec:
  scaleTargetRef:
    name: trino-worker
  minReplicaCount: 1
  maxReplicaCount: 5
...
  triggers:
    - type: prometheus
      metricType: Value
      metadata:
        serverAddress: "http://prometheus.example.com"
        threshold: "1"
        metricName: queued_queries
        query: sum by (job) (avg_over_time(trino_queued_queries{job="trino"}[30s]))
        authModes: "basic"
...
```

Thank you!

For your time and attention.