package com.twitter.product\_mixer.core.service.component\_registry

import com.twitter.finagle.stats.StatsReceiver

import com.twitter.product\_mixer.core.model.common.Component

import com.twitter.product\_mixer.core.model.common.identifier.ComponentIdentifier

import com.twitter.product\_mixer.core.model.common.identifier.ComponentIdentifierStack

import com.twitter.util.Activity

import com.twitter.util.Future

import com.twitter.util.Try

import com.twitter.util.logging.Logging

import java.util.concurrent.ConcurrentHashMap

import javax.inject.Inject

import javax.inject.Singleton

import scala.collection.JavaConverters.\_

/\*\*

\* The [[ComponentRegistry]] works closely with [[ComponentIdentifier]]s and the [[com.twitter.product\_mixer.core.product.registry.ProductPipelineRegistry]]

\* to provide the Product Mixer framework information about the [[com.twitter.product\_mixer.core.pipeline.Pipeline]]s and [[Component]]s

\* that make up an application.

\*

\* This registration allows us to configure alerts and dashboards,

\* to query your application structure letting us display the graph of the execution and the results of queries,

\* and to garner insight into usages.

\*

\* The registry is a snapshot of the state of the world when pipelines were last built successfully.

\* For most services, this only happens once on startup. However, some services may rebuild their

\* pipelines dynamically later on.

\*/

@Singleton

class ComponentRegistry @Inject() (statsReceiver: StatsReceiver) {

// Initially pending until the first snapshot is built by [[ProductPipelineRegistry]]

private val (snapshotActivity, snapshotWitness) = Activity[ComponentRegistrySnapshot]()

private val snapshotCount = statsReceiver.counter("ComponentRegistry", "SnapshotCount")

def get: Future[ComponentRegistrySnapshot] = snapshotActivity.values.toFuture.lowerFromTry

private[core] def set(snapshot: ComponentRegistrySnapshot): Unit = {

snapshotCount.incr()

snapshotWitness.notify(Try(snapshot))

}

}

class ComponentRegistrySnapshot() extends Logging {

/\*\* for storing the [[RegisteredComponent]]s \*/

private[this] val componentRegistry =

new ConcurrentHashMap[ComponentIdentifier, RegisteredComponent]

/\*\* for determining the children of a [[ComponentIdentifier]] \*/

private[this] val componentChildren =

new ConcurrentHashMap[ComponentIdentifier, Set[ComponentIdentifier]]

/\*\* for determining [[ComponentIdentifier]] uniqueness within a given [[ComponentIdentifierStack]] \*/

private[this] val componentHierarchy =

new ConcurrentHashMap[ComponentIdentifierStack, Set[ComponentIdentifier]]

/\*\*

\* Register the given [[Component]] at the end of path provided by `parentIdentifierStack`

\* or throws an exception if adding the component results in an invalid configuration.

\*

\* @throws ChildComponentCollisionException if a [[Component]] with the same [[ComponentIdentifier]] is registered

\* more than once under the same parent.

\* e.g. if you register `ComponentA` under `ProductA -> PipelineA` twice,

\* this exception will be thrown when registering `ComponentA` the second

\* time. This is pretty much always a configuration error due to copy-pasting

\* and forgetting to update the identifier, or accidentally using the same

\* component twice under the same parent. If this didn't throw, stats from

\* these 2 components would be indistinguishable.

\*

\* @throws ComponentIdentifierCollisionException if a [[Component]] with the same [[ComponentIdentifier]] is registered

\* but it's type is not the same as a previously registered [[Component]]

\* with the same [[ComponentIdentifier]]

\* e.g. if you register 2 [[Component]]s with the same [[ComponentIdentifier]]

\* such as `new Component` and an instance of

\* `class MyComponent extends Component` the `new Component` will have a

\* type of `Component` and the other one will have a type of `MyComponent`

\* which will throw. This is usually due to copy-pasting a component as

\* a starting point and forgetting to update the identifier. If this

\* didn't throw, absolute stats from these 2 components would be

\* indistinguishable.

\*

\*

\* @note this will log details of component identifier reuse if the underling components are not equal, but otherwise are of the same class.

\* Their stats will be merged and indistinguishable but since they are the same name and same class, we assume the differences are

\* minor enough that this is okay, but make a note in the log at startup in case someone sees unexpected metrics, we can look

\* back at the logs and see the details.

\*

\* @param component the component to register

\* @param parentIdentifierStack the complete [[ComponentIdentifierStack]] excluding the current [[Component]]'s [[ComponentIdentifier]]

\*/

def register(

component: Component,

parentIdentifierStack: ComponentIdentifierStack

): Unit = synchronized {

val identifier = component.identifier

val parentIdentifier = parentIdentifierStack.peek

val registeredComponent =

RegisteredComponent(identifier, component, component.identifier.file.value)

componentRegistry.asScala

.get(identifier)

.filter(\_.component != component) // only do the foreach if the components aren't equal

.foreach {

case existingComponent if existingComponent.component.getClass != component.getClass =>

/\*\*

\* The same component may be registered under different parent components.

\* However, different component types cannot use the same component identifier.

\*

\* This catches some copy-pasting of a config or component and forgetting to update the identifier.

\*/

throw new ComponentIdentifierCollisionException(

componentIdentifier = identifier,

component = registeredComponent,

existingComponent = componentRegistry.get(identifier),

parentIdentifierStack = parentIdentifierStack,

existingIdentifierStack = componentHierarchy.search[ComponentIdentifierStack](

1,

(stack, identifiers) => if (identifiers.contains(identifier)) stack else null)

)

case existingComponent =>

/\*\*

\* The same component may be registered under different parent components.

\* However, if the components are not equal it \_\_may be\_\_ a configuration error

\* so we log a detailed description of the issue in case they need to debug.

\*

\* This warns customers of some copy-pasting of a config or component and forgetting to update the

\* identifier and of reusing components with hard-coded values which are configured differently.

\*/

val existingIdentifierStack = componentHierarchy.search[ComponentIdentifierStack](

1,

(stack, identifiers) => if (identifiers.contains(identifier)) stack else null)

logger.info(

s"Found duplicate identifiers for non-equal components, $identifier from ${registeredComponent.sourceFile} " +

s"under ${parentIdentifierStack.componentIdentifiers.reverse.mkString(" -> ")} " +

s"was already defined and is unequal to ${existingComponent.sourceFile} " +

s"under ${existingIdentifierStack.componentIdentifiers.reverse.mkString(" -> ")}. " +

s"Merging these components in the registry, this will result in their metrics being merged. " +

s"If these components should have separate metrics, consider providing unique identifiers for them instead."

)

}

/\*\* The same component may not be registered multiple times under the same parent \*/

if (componentHierarchy.getOrDefault(parentIdentifierStack, Set.empty).contains(identifier))

throw new ChildComponentCollisionException(identifier, parentIdentifierStack)

// add component to registry

componentRegistry.putIfAbsent(identifier, registeredComponent)

// add component to parent's `children` set for easy lookup

componentChildren.merge(parentIdentifier, Set(identifier), \_ ++ \_)

// add the component to the hierarchy under it's parent's identifier stack

componentHierarchy.merge(parentIdentifierStack, Set(identifier), \_ ++ \_)

}

def getAllRegisteredComponents: Seq[RegisteredComponent] =

componentRegistry.values.asScala.toSeq.sorted

def getChildComponents(component: ComponentIdentifier): Seq[ComponentIdentifier] =

Option(componentChildren.get(component)) match {

case Some(components) => components.toSeq.sorted(ComponentIdentifier.ordering)

case None => Seq.empty

}

}

class ComponentIdentifierCollisionException(

componentIdentifier: ComponentIdentifier,

component: RegisteredComponent,

existingComponent: RegisteredComponent,

parentIdentifierStack: ComponentIdentifierStack,

existingIdentifierStack: ComponentIdentifierStack)

extends IllegalArgumentException(

s"Tried to register component $componentIdentifier: of type ${component.component.getClass} from ${component.sourceFile} " +

s"under ${parentIdentifierStack.componentIdentifiers.reverse.mkString(" -> ")} " +

s"but it was already defined with a different type ${existingComponent.component.getClass} from ${existingComponent.sourceFile} " +

s"under ${existingIdentifierStack.componentIdentifiers.reverse.mkString(" -> ")}. " +

s"Ensure you aren't reusing a component identifier which can happen when copy-pasting existing component code by accident")

class ChildComponentCollisionException(

componentIdentifier: ComponentIdentifier,

parentIdentifierStack: ComponentIdentifierStack)

extends IllegalArgumentException(

s"Component $componentIdentifier already defined under parent component $parentIdentifierStack")