To create a **Dashboard Stack** that displays **ALB (Application Load Balancer) metrics** in **CloudWatch**, follow these steps:

**1. Create a New Stack File (dashboard\_stack.ts)**

Create a new file **dashboard\_stack.ts** inside your **stacks** directory.

**2. Import Required AWS CDK Modules**

You'll need AWS CDK modules for CloudWatch and ECS:

typescript

CopyEdit

import \* as cdk from 'aws-cdk-lib';

import \* as cloudwatch from 'aws-cdk-lib/aws-cloudwatch';

import \* as elbv2 from 'aws-cdk-lib/aws-elasticloadbalancingv2';

import { Construct } from 'constructs';

export class DashboardStack extends cdk.Stack {

constructor(scope: Construct, id: string, props?: cdk.StackProps) {

super(scope, id, props);

// Import existing ALB by name

const loadBalancer = elbv2.ApplicationLoadBalancer.fromLookup(this, 'ImportedALB', {

loadBalancerTags: {

Name: 'YourALBName', // Replace with actual ALB name

},

});

// Create a new CloudWatch Dashboard

const dashboard = new cloudwatch.Dashboard(this, 'ALBDashboard', {

dashboardName: 'ALB-Monitoring-Dashboard',

});

// Add ALB Metrics to the Dashboard

const albMetrics = [

{

title: 'Request Count',

metric: loadBalancer.metricRequestCount(),

},

{

title: 'Active Connections',

metric: loadBalancer.metricActiveConnectionCount(),

},

{

title: 'Target Response Time',

metric: loadBalancer.metricTargetResponseTime(),

},

{

title: 'HTTP 4XX Errors',

metric: loadBalancer.metricHttpCodeElb(elbv2.HttpCodeElb.ELB\_4XX\_COUNT),

},

{

title: 'HTTP 5XX Errors',

metric: loadBalancer.metricHttpCodeElb(elbv2.HttpCodeElb.ELB\_5XX\_COUNT),

},

];

// Add Widgets to Dashboard

albMetrics.forEach((albMetric, index) => {

dashboard.addWidgets(

new cloudwatch.GraphWidget({

title: albMetric.title,

left: [albMetric.metric],

width: 12, // Half-width

})

);

});

new cdk.CfnOutput(this, 'DashboardURL', {

value: `https://console.aws.amazon.com/cloudwatch/home?region=${this.region}#dashboards:name=ALB-Monitoring-Dashboard`,

description: 'CloudWatch Dashboard URL',

});

}

}

**3. Update bin/app.ts to Include DashboardStack**

Modify your **bin/app.ts** to deploy this new stack:

typescript

CopyEdit

import \* as cdk from 'aws-cdk-lib';

import { AlbStack } from '../lib/alb\_stack';

import { DashboardStack } from '../lib/dashboard\_stack';

const app = new cdk.App();

const albStack = new AlbStack(app, 'AlbStack');

new DashboardStack(app, 'DashboardStack', {

env: { region: 'us-east-1' }, // Replace with your region

});

**4. Deploy the Stack**

Run the following CDK commands:

sh

CopyEdit

cdk synth

cdk deploy

**5. Verify the Dashboard**

1. Go to the **AWS CloudWatch Console**.
2. Navigate to **Dashboards**.
3. Find **ALB-Monitoring-Dashboard** and open it.

**Features of This Setup**

✅ Automatically pulls ALB metrics.  
✅ Displays **Request Count, Active Connections, Target Response Time, HTTP 4XX & 5XX Errors**.  
✅ Provides a CloudWatch URL as an output after deployment.

Let me know if you need modifications!

4o

**You said:**

how the alb defined in other stack is referred in this cloudwatch stack

**ChatGPT said:**

ChatGPT

Since the **ALB (Application Load Balancer)** is created in AlbStack, you need a way to reference it in the DashboardStack. There are two main approaches to achieve this:

**1️⃣ Pass the ALB as a Parameter to DashboardStack**

Modify AlbStack to expose the ALB instance, then pass it to DashboardStack.

**Step 1: Modify alb\_stack.ts to Export ALB**

Update AlbStack to expose the ALB as a class property.

typescript

CopyEdit

import \* as cdk from 'aws-cdk-lib';

import \* as elbv2 from 'aws-cdk-lib/aws-elasticloadbalancingv2';

import \* as ec2 from 'aws-cdk-lib/aws-ec2';

import { Construct } from 'constructs';

export class AlbStack extends cdk.Stack {

public readonly loadBalancer: elbv2.ApplicationLoadBalancer;

constructor(scope: Construct, id: string, props?: cdk.StackProps) {

super(scope, id, props);

const vpc = new ec2.Vpc(this, 'VPC', { maxAzs: 2 });

this.loadBalancer = new elbv2.ApplicationLoadBalancer(this, 'MyALB', {

vpc,

internetFacing: true,

});

new cdk.CfnOutput(this, 'ALBArn', {

value: this.loadBalancer.loadBalancerArn,

});

}

}

**Step 2: Modify dashboard\_stack.ts to Accept ALB**

Modify DashboardStack to receive the ALB instance via constructor parameters.

typescript

CopyEdit

import \* as cdk from 'aws-cdk-lib';

import \* as cloudwatch from 'aws-cdk-lib/aws-cloudwatch';

import \* as elbv2 from 'aws-cdk-lib/aws-elasticloadbalancingv2';

import { Construct } from 'constructs';

interface DashboardStackProps extends cdk.StackProps {

loadBalancer: elbv2.ApplicationLoadBalancer;

}

export class DashboardStack extends cdk.Stack {

constructor(scope: Construct, id: string, props: DashboardStackProps) {

super(scope, id, props);

const loadBalancer = props.loadBalancer;

const dashboard = new cloudwatch.Dashboard(this, 'ALBDashboard', {

dashboardName: 'ALB-Monitoring-Dashboard',

});

const albMetrics = [

{ title: 'Request Count', metric: loadBalancer.metricRequestCount() },

{ title: 'Active Connections', metric: loadBalancer.metricActiveConnectionCount() },

{ title: 'Target Response Time', metric: loadBalancer.metricTargetResponseTime() },

{ title: 'HTTP 4XX Errors', metric: loadBalancer.metricHttpCodeElb(elbv2.HttpCodeElb.ELB\_4XX\_COUNT) },

{ title: 'HTTP 5XX Errors', metric: loadBalancer.metricHttpCodeElb(elbv2.HttpCodeElb.ELB\_5XX\_COUNT) },

];

albMetrics.forEach((albMetric) => {

dashboard.addWidgets(

new cloudwatch.GraphWidget({

title: albMetric.title,

left: [albMetric.metric],

width: 12,

})

);

});

new cdk.CfnOutput(this, 'DashboardURL', {

value: `https://console.aws.amazon.com/cloudwatch/home?region=${this.region}#dashboards:name=ALB-Monitoring-Dashboard`,

description: 'CloudWatch Dashboard URL',

});

}

}

**Step 3: Modify bin/app.ts to Pass ALB**

Now, update bin/app.ts to pass the ALB reference from AlbStack to DashboardStack.

typescript

CopyEdit

import \* as cdk from 'aws-cdk-lib';

import { AlbStack } from '../lib/alb\_stack';

import { DashboardStack } from '../lib/dashboard\_stack';

const app = new cdk.App();

const albStack = new AlbStack(app, 'AlbStack');

new DashboardStack(app, 'DashboardStack', {

env: { region: 'us-east-1' },

loadBalancer: albStack.loadBalancer,

});

**Now Deploy the Stacks**

sh

CopyEdit

cdk synth

cdk deploy

**2️⃣ Use fromLookup to Import ALB**

If AlbStack and DashboardStack are deployed separately (or in different CDK apps), you can **import the ALB using fromLookup** instead of passing it.

**Modify dashboard\_stack.ts to Import ALB**

typescript

CopyEdit

const loadBalancer = elbv2.ApplicationLoadBalancer.fromLookup(this, 'ImportedALB', {

loadBalancerTags: {

Name: 'YourALBName', // Replace with the ALB Name from `AlbStack`

},

});

This approach is useful if you don't want to modify AlbStack, but it requires that the ALB already exists in AWS.

**Which Approach to Use?**

✅ **Approach 1 (Pass as Parameter)** → Recommended when both stacks are in the same CDK app.  
✅ **Approach 2 (fromLookup)** → Useful when AlbStack is deployed separately or in another CDK project.

Let me know if you need further clarification! 🚀

4o

**You said:**

there are many cdk stack codes in single cloud deployment. ALB is defined in one cdk stack code in ts and similary s3 , rds and others are defined as sepaater ts stack file. buit they AR EPART OF ONE APPLICAITON. I think th elookup mehtod is better

**ChatGPT said:**

ChatGPT

Yes! Since all the CDK stacks (ALB, S3, RDS, etc.) are part of the same **cloud deployment but defined in separate stack files**, using **fromLookup** is the better approach.

This way, your DashboardStack can fetch the **ALB ARN dynamically**, even if it's deployed separately.

**✅ Using fromLookup to Reference ALB in DashboardStack**

Modify your dashboard\_stack.ts file to import the ALB dynamically.

**Step 1: Import Required Modules**

In **dashboard\_stack.ts**, import the necessary AWS CDK libraries:

typescript

CopyEdit

import \* as cdk from 'aws-cdk-lib';

import \* as cloudwatch from 'aws-cdk-lib/aws-cloudwatch';

import \* as elbv2 from 'aws-cdk-lib/aws-elasticloadbalancingv2';

import { Construct } from 'constructs';

**Step 2: Use fromLookup to Get ALB**

Modify the DashboardStack constructor to retrieve the **ALB ARN dynamically**:

typescript

CopyEdit

export class DashboardStack extends cdk.Stack {

constructor(scope: Construct, id: string, props?: cdk.StackProps) {

super(scope, id, props);

// Look up ALB using its name (ensure ALB has a Name tag in `AlbStack`)

const loadBalancer = elbv2.ApplicationLoadBalancer.fromLookup(this, 'ImportedALB', {

loadBalancerTags: { Name: 'YourALBName' }, // 🔹 Replace 'YourALBName' with the actual ALB name

});

// Create a CloudWatch Dashboard

const dashboard = new cloudwatch.Dashboard(this, 'ALBDashboard', {

dashboardName: 'ALB-Monitoring-Dashboard',

});

// Add ALB Metrics

const albMetrics = [

{ title: 'Request Count', metric: loadBalancer.metricRequestCount() },

{ title: 'Active Connections', metric: loadBalancer.metricActiveConnectionCount() },

{ title: 'Target Response Time', metric: loadBalancer.metricTargetResponseTime() },

{ title: 'HTTP 4XX Errors', metric: loadBalancer.metricHttpCodeElb(elbv2.HttpCodeElb.ELB\_4XX\_COUNT) },

{ title: 'HTTP 5XX Errors', metric: loadBalancer.metricHttpCodeElb(elbv2.HttpCodeElb.ELB\_5XX\_COUNT) },

];

albMetrics.forEach((albMetric) => {

dashboard.addWidgets(

new cloudwatch.GraphWidget({

title: albMetric.title,

left: [albMetric.metric],

width: 12,

})

);

});

// Print Dashboard URL as Output

new cdk.CfnOutput(this, 'DashboardURL', {

value: `https://console.aws.amazon.com/cloudwatch/home?region=${this.region}#dashboards:name=ALB-Monitoring-Dashboard`,

description: 'CloudWatch Dashboard URL',

});

}

}

**Step 3: Ensure ALB is Tagged in AlbStack**

In **alb\_stack.ts**, make sure the ALB is tagged properly for lookup:

typescript

CopyEdit

const alb = new elbv2.ApplicationLoadBalancer(this, 'MyALB', {

vpc,

internetFacing: true,

});

// Add a Name tag (IMPORTANT for `fromLookup` to work)

cdk.Tags.of(alb).add('Name', 'YourALBName'); // 🔹 Use the same name in `fromLookup`

**Step 4: Deploy the Stacks**

Now, **deploy all stacks**, and the DashboardStack will automatically find the ALB.

sh

CopyEdit

cdk synth

cdk deploy

**✅ Why is fromLookup Better for Your Use Case?**

* 🔹 **No need to pass ALB between stacks**
* 🔹 **Works even if stacks are deployed separately**
* 🔹 **Can be used across multiple CDK apps**
* 🔹 **Keeps stacks loosely coupled**

Let me know if you need any refinements! 🚀

4o

**You said:**

I will add more widgets (graphs) fro difference AWS services sim=milar to alb i.e S3, API g/w, ECS fargate services as separate widgets, bu tthey can be in the same dashbvoard, If thats the case, how other stacks shoul dtag themselbes and how this dashboard will refer them iw hile creating widgets for those services defined in separate stasks in same project

**ChatGPT said:**

ChatGPT

Since you plan to add widgets for **S3, API Gateway, ECS Fargate, and other AWS services** to the same **CloudWatch Dashboard**, you need a **consistent tagging strategy** across your CDK stacks.

**✅ How Should Other Stacks Tag Themselves?**

Each stack (S3, API Gateway, ECS Fargate, etc.) should **tag its resources with a unique "Name"**. This allows the DashboardStack to **dynamically discover and reference them** using fromLookup.

**Example Tagging Strategy in Each Stack**

Modify each stack to **tag the primary AWS resource** properly.

**🔹 ALB Stack (alb\_stack.ts)**

typescript

CopyEdit

const alb = new elbv2.ApplicationLoadBalancer(this, 'MyALB', {

vpc,

internetFacing: true,

});

cdk.Tags.of(alb).add('Name', 'MyApplicationALB'); // 🔹 Important!

**🔹 S3 Stack (s3\_stack.ts)**

typescript

CopyEdit

const bucket = new s3.Bucket(this, 'MyS3Bucket', {

bucketName: 'my-app-bucket',

});

cdk.Tags.of(bucket).add('Name', 'MyAppS3Bucket'); // 🔹 Important!

**🔹 API Gateway Stack (api\_gateway\_stack.ts)**

typescript

CopyEdit

const apiGateway = new apigateway.RestApi(this, 'MyApiGateway', {

restApiName: 'MyAppApiGateway',

});

cdk.Tags.of(apiGateway).add('Name', 'MyAppApiGateway'); // 🔹 Important!

**🔹 ECS Fargate Stack (ecs\_stack.ts)**

typescript

CopyEdit

const ecsCluster = new ecs.Cluster(this, 'MyEcsCluster', {

vpc,

clusterName: 'MyAppECSCluster',

});

cdk.Tags.of(ecsCluster).add('Name', 'MyAppECSCluster'); // 🔹 Important!

**✅ How DashboardStack References These Resources**

In dashboard\_stack.ts, use fromLookup (or equivalent) to retrieve the resources by their **tagged Name**.

typescript

CopyEdit

import \* as cdk from 'aws-cdk-lib';

import \* as cloudwatch from 'aws-cdk-lib/aws-cloudwatch';

import \* as elbv2 from 'aws-cdk-lib/aws-elasticloadbalancingv2';

import \* as s3 from 'aws-cdk-lib/aws-s3';

import \* as apigateway from 'aws-cdk-lib/aws-apigateway';

import \* as ecs from 'aws-cdk-lib/aws-ecs';

import { Construct } from 'constructs';

export class DashboardStack extends cdk.Stack {

constructor(scope: Construct, id: string, props?: cdk.StackProps) {

super(scope, id, props);

// ✅ Lookup ALB

const loadBalancer = elbv2.ApplicationLoadBalancer.fromLookup(this, 'ImportedALB', {

loadBalancerTags: { Name: 'MyApplicationALB' },

});

// ✅ Lookup S3 Bucket

const bucket = s3.Bucket.fromBucketName(this, 'ImportedS3Bucket', 'my-app-bucket');

// ✅ Lookup API Gateway

const apiGateway = apigateway.RestApi.fromRestApiAttributes(this, 'ImportedApiGateway', {

restApiId: 'YOUR\_API\_GATEWAY\_ID', // Replace with actual ID if needed

rootResourceId: 'YOUR\_API\_GATEWAY\_ROOT\_RESOURCE\_ID',

});

// ✅ Lookup ECS Cluster

const ecsCluster = ecs.Cluster.fromClusterAttributes(this, 'ImportedEcsCluster', {

clusterName: 'MyAppECSCluster',

vpc: undefined, // Specify the VPC if required

});

// ✅ Create CloudWatch Dashboard

const dashboard = new cloudwatch.Dashboard(this, 'AppMonitoringDashboard', {

dashboardName: 'Application-Monitoring-Dashboard',

});

// ✅ Add ALB Widgets

dashboard.addWidgets(

new cloudwatch.GraphWidget({

title: 'ALB Request Count',

left: [loadBalancer.metricRequestCount()],

width: 12,

}),

new cloudwatch.GraphWidget({

title: 'ALB HTTP 4XX Errors',

left: [loadBalancer.metricHttpCodeElb(elbv2.HttpCodeElb.ELB\_4XX\_COUNT)],

width: 12,

})

);

// ✅ Add S3 Widgets

dashboard.addWidgets(

new cloudwatch.GraphWidget({

title: 'S3 Bucket Size',

left: [bucket.metric('BucketSizeBytes', { statistic: 'Average' })],

width: 12,

}),

new cloudwatch.GraphWidget({

title: 'S3 Number of Objects',

left: [bucket.metric('NumberOfObjects', { statistic: 'Average' })],

width: 12,

})

);

// ✅ Add API Gateway Widgets

dashboard.addWidgets(

new cloudwatch.GraphWidget({

title: 'API Gateway Request Count',

left: [apiGateway.metricCount()],

width: 12,

}),

new cloudwatch.GraphWidget({

title: 'API Gateway 5XX Errors',

left: [apiGateway.metricServerError()],

width: 12,

})

);

// ✅ Add ECS Widgets

dashboard.addWidgets(

new cloudwatch.GraphWidget({

title: 'ECS Running Tasks',

left: [ecsCluster.metric('RunningTaskCount')],

width: 12,

}),

new cloudwatch.GraphWidget({

title: 'ECS CPU Utilization',

left: [ecsCluster.metric('CPUUtilization')],

width: 12,

})

);

// ✅ Print Dashboard URL as Output

new cdk.CfnOutput(this, 'DashboardURL', {

value: `https://console.aws.amazon.com/cloudwatch/home?region=${this.region}#dashboards:name=Application-Monitoring-Dashboard`,

description: 'CloudWatch Dashboard URL',

});

}

}

**✅ Summary of Approach**

| **AWS Service** | **How to Reference in DashboardStack** |
| --- | --- |
| **ALB** | fromLookup() with Name tag |
| **S3** | fromBucketName() |
| **API Gateway** | fromRestApiAttributes() (with API ID) |
| **ECS Cluster** | fromClusterAttributes() (with cluster name) |

**🚀 Benefits of This Approach**

✔ **No Need to Pass Resource References Manually**  
✔ **Stacks Remain Independent** (Loose Coupling)  
✔ **Works Across Multi-Stack Deployments**  
✔ **Easily Extendable to Other AWS Services**

Would this work for you? Let me know if you need adjustments! 🚀🔥

4o

**You said:**

whast the right method to identify services such as rds mssql db, ecs fargate services (i.e 4 diff services), MSK, redis, ALB, API Gateway, and modify the dashboard correctly to habdle them

**ChatGPT said:**

ChatGPT

Since you have multiple AWS services (**RDS MSSQL, ECS Fargate, MSK (Kafka), Redis, ALB, API Gateway**) and want to add them dynamically to **CloudWatch Dashboards**, the best approach is:

1. **Use fromLookup() or equivalent methods** to reference services.
2. **Ensure each service is tagged properly** (e.g., Name: MyAppRDS).
3. **Modify dashboard\_stack.ts to dynamically discover these services** and create **Graph Widgets** for them.

**✅ How to Identify Each AWS Service in CDK**

Below are the **best ways** to fetch these resources dynamically:

| **AWS Service** | **How to Reference in CDK Dashboard** |
| --- | --- |
| **RDS (MSSQL)** | rds.DatabaseInstance.fromDatabaseInstanceAttributes() |
| **ECS Fargate** | ecs.Cluster.fromClusterAttributes() |
| **Amazon MSK (Kafka)** | msk.Cluster.fromClusterArn() |
| **Redis (Elasticache)** | elasticache.CfnCacheCluster.fromCacheClusterAttributes() |
| **ALB (Application Load Balancer)** | elbv2.ApplicationLoadBalancer.fromLookup() |
| **API Gateway** | apigateway.RestApi.fromRestApiAttributes() |

**✅ Step 1: Tag Resources in Their Respective Stacks**

Each stack should **tag its primary AWS resource**, so dashboard\_stack.ts can **discover** them.

**RDS MSSQL Stack (rds\_stack.ts)**

typescript

CopyEdit

const rdsInstance = new rds.DatabaseInstance(this, 'MyRDS', {

engine: rds.DatabaseInstanceEngine.sqlServerEx({

version: rds.SqlServerEngineVersion.VER\_15,

}),

vpc,

instanceType: ec2.InstanceType.of(ec2.InstanceClass.BURSTABLE3, ec2.InstanceSize.MICRO),

});

cdk.Tags.of(rdsInstance).add('Name', 'MyAppRDS'); // 🔹 Important!

**ECS Fargate Stack (ecs\_stack.ts)**

typescript

CopyEdit

const ecsCluster = new ecs.Cluster(this, 'MyEcsCluster', {

vpc,

clusterName: 'MyAppECSCluster',

});

cdk.Tags.of(ecsCluster).add('Name', 'MyAppECSCluster'); // 🔹 Important!

**Amazon MSK (Kafka) Stack (msk\_stack.ts)**

typescript

CopyEdit

const mskCluster = new msk.CfnCluster(this, 'MyMSKCluster', {

clusterName: 'MyAppMSK',

kafkaVersion: '3.2.0',

numberOfBrokerNodes: 3,

});

cdk.Tags.of(mskCluster).add('Name', 'MyAppMSK'); // 🔹 Important!

**Redis (ElastiCache) Stack (redis\_stack.ts)**

typescript

CopyEdit

const redisCluster = new elasticache.CfnCacheCluster(this, 'MyRedis', {

cacheNodeType: 'cache.t3.micro',

engine: 'redis',

numCacheNodes: 1,

});

cdk.Tags.of(redisCluster).add('Name', 'MyAppRedis'); // 🔹 Important!

**ALB Stack (alb\_stack.ts)**

typescript

CopyEdit

const alb = new elbv2.ApplicationLoadBalancer(this, 'MyALB', {

vpc,

internetFacing: true,

});

cdk.Tags.of(alb).add('Name', 'MyApplicationALB'); // 🔹 Important!

**API Gateway Stack (api\_gateway\_stack.ts)**

typescript

CopyEdit

const apiGateway = new apigateway.RestApi(this, 'MyApiGateway', {

restApiName: 'MyAppApiGateway',

});

cdk.Tags.of(apiGateway).add('Name', 'MyAppApiGateway'); // 🔹 Important!

**✅ Step 2: Modify dashboard\_stack.ts to Discover and Monitor Services**

Now, we create **Graph Widgets** for each service.

typescript

CopyEdit

import \* as cdk from 'aws-cdk-lib';

import \* as cloudwatch from 'aws-cdk-lib/aws-cloudwatch';

import \* as elbv2 from 'aws-cdk-lib/aws-elasticloadbalancingv2';

import \* as s3 from 'aws-cdk-lib/aws-s3';

import \* as apigateway from 'aws-cdk-lib/aws-apigateway';

import \* as ecs from 'aws-cdk-lib/aws-ecs';

import \* as rds from 'aws-cdk-lib/aws-rds';

import \* as msk from 'aws-cdk-lib/aws-msk';

import \* as elasticache from 'aws-cdk-lib/aws-elasticache';

import { Construct } from 'constructs';

export class DashboardStack extends cdk.Stack {

constructor(scope: Construct, id: string, props?: cdk.StackProps) {

super(scope, id, props);

// ✅ Lookup ALB

const loadBalancer = elbv2.ApplicationLoadBalancer.fromLookup(this, 'ImportedALB', {

loadBalancerTags: { Name: 'MyApplicationALB' },

});

// ✅ Lookup RDS MSSQL

const rdsInstance = rds.DatabaseInstance.fromDatabaseInstanceAttributes(this, 'ImportedRDS', {

instanceIdentifier: 'MyAppRDS',

});

// ✅ Lookup ECS Cluster

const ecsCluster = ecs.Cluster.fromClusterAttributes(this, 'ImportedEcsCluster', {

clusterName: 'MyAppECSCluster',

});

// ✅ Lookup MSK Cluster

const mskCluster = msk.Cluster.fromClusterArn(this, 'ImportedMSK', `arn:aws:kafka:${this.region}:${this.account}:cluster/MyAppMSK/...`);

// ✅ Lookup Redis Cluster

const redisCluster = elasticache.CfnCacheCluster.fromCacheClusterAttributes(this, 'ImportedRedis', {

cacheClusterId: 'MyAppRedis',

});

// ✅ Lookup API Gateway

const apiGateway = apigateway.RestApi.fromRestApiAttributes(this, 'ImportedApiGateway', {

restApiId: 'YOUR\_API\_GATEWAY\_ID', // Replace with actual ID if needed

rootResourceId: 'YOUR\_API\_GATEWAY\_ROOT\_RESOURCE\_ID',

});

// ✅ Create CloudWatch Dashboard

const dashboard = new cloudwatch.Dashboard(this, 'AppMonitoringDashboard', {

dashboardName: 'Application-Monitoring-Dashboard',

});

// ✅ Add ALB Widgets

dashboard.addWidgets(new cloudwatch.GraphWidget({

title: 'ALB Request Count',

left: [loadBalancer.metricRequestCount()],

width: 12,

}));

// ✅ Add RDS Widgets

dashboard.addWidgets(new cloudwatch.GraphWidget({

title: 'RDS CPU Utilization',

left: [rdsInstance.metricCPUUtilization()],

width: 12,

}));

// ✅ Add ECS Widgets

dashboard.addWidgets(new cloudwatch.GraphWidget({

title: 'ECS Running Tasks',

left: [ecsCluster.metric('RunningTaskCount')],

width: 12,

}));

// ✅ Add MSK Widgets

dashboard.addWidgets(new cloudwatch.GraphWidget({

title: 'MSK Active Connections',

left: [mskCluster.metric('Kafka.ActiveConnections')],

width: 12,

}));

// ✅ Add Redis Widgets

dashboard.addWidgets(new cloudwatch.GraphWidget({

title: 'Redis CPU Utilization',

left: [redisCluster.metric('CPUUtilization')],

width: 12,

}));

// ✅ Add API Gateway Widgets

dashboard.addWidgets(new cloudwatch.GraphWidget({

title: 'API Gateway Request Count',

left: [apiGateway.metricCount()],

width: 12,

}));

// ✅ Print Dashboard URL as Output

new cdk.CfnOutput(this, 'DashboardURL', {

value: `https://console.aws.amazon.com/cloudwatch/home?region=${this.region}#dashboards:name=Application-Monitoring-Dashboard`,

description: 'CloudWatch Dashboard URL',

});

}

}

**🚀 Summary of Approach**

1. **Tag Resources Properly** (Name: MyAppRDS, Name: MyAppECSCluster, etc.).
2. **Use fromLookup() or equivalent in dashboard\_stack.ts**.
3. **Dynamically create widgets for each AWS service**.

**🔥 Benefits**

✔ **No manual passing of resource references**  
✔ **Stacks remain independent (Loose Coupling)**  
✔ **Easily extendable (Just add new tags & metrics!)**  
✔ **Single Dashboard for all services**

Would this work for you? Let me know if you need modifications! 🚀🔥