# **Multi-Tenant System for Sales and Distribution**

**Amazon Web Service**

Amazon is a IAAS infrastructure as a service that provides virtualized cloud computing resources over the internet.

To use amazon relational database, we need AWS elastic Beanstalk.

AWS Elastic Beanstalk provides support for [running Amazon Relational Database Service Amazon RDS](https://docs.aws.amazon.com/elasticbeanstalk/latest/dg/using-features.managing.db.html)

To decouple your database instance from your environment, you can run a database instance in Amazon RDS and configure your application to connect to it on launch. This enables you to connect multiple environments to a database, terminate an environment without affecting the database

**AWS Elastic Beanstlk environment configuration for RDBS**

Here are few steps you can follow to configure environment properties for AWS relational database

1. Open the [Elastic Beanstalk console](https://console.aws.amazon.com/elasticbeanstalk).
2. Navigate to the [management page](https://docs.aws.amazon.com/elasticbeanstalk/latest/dg/environments-console.html) for your environment.
3. Choose Configuration.
4. On the Software configuration card, choose Modify.
5. In the Environment properties section, define the variables that your application reads to construct a connection string. For compatibility with environments that have an integrated RDS DB instance, use the following.
   * **RDS\_HOSTNAME** – The hostname of the DB instance.

Amazon RDS console label – **Endpoint** (this is the hostname)

* + **RDS\_PORT** – The port on which the DB instance accepts connections. The default value varies among DB engines.

Amazon RDS console label – **Port**

* + **RDS\_DB\_NAME** – The database name, **ebdb**.

Amazon RDS console label – **DB Name**

* + **RDS\_USERNAME** – The user name that you configured for your database.

Amazon RDS console label – **Username**

* + **RDS\_PASSWORD** – The password that you configured for your database.


        Environment Properties section with RDS properties added
      

# Adding an Amazon RDS DB Instance to your Node.js Application Environment

Downloading driver

{

"name": "my-app",

"version": "0.0.1",

"private": true,

"dependencies": {

"ejs": "latest",

"aws-sdk": "latest",

"express": "latest",

"body-parser": "latest",

**"mssql": "latest"**

},

"scripts": {

"start": "node app.js"

}

}

**Common Driver Packages for Node.js**

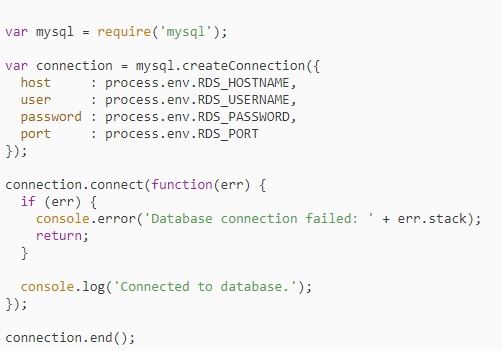
* **MySQL** – mysql
* **PostgreSQL** – pg
* **SQL Server** – mssql
* **Oracle** – oracle or oracledb

The Oracle package and version depend on the Node.js version you're using:

* + **Node.js 6.x, 8.x** – Use the latest version of oracledb.
  + **Node.js 4.x** – Use the oracledb version 2.2.0.
  + **Node.js 5.x, 7.x** – Use the latest version of oracle. The oracledb package doesn't support these Node.js versions.

## Connecting to a Database

Elastic Beanstalk provides connection information for attached DB instances in environment properties. Use os.environ['*VARIABLE*'] to read the properties and configure a database connection.



# Adding an Amazon RDS DB Instance to Your .NET Application Environment

## Downloading a Driver

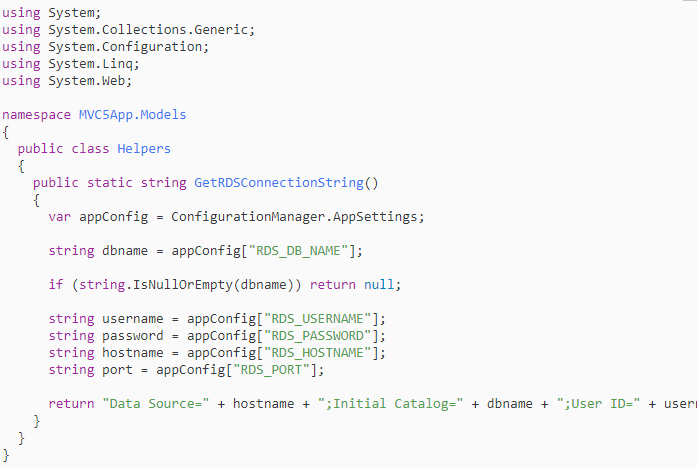
Download and install the EntityFramework package and a database driver for your development environment with NuGet.

**Common Entity Framework Database Providers for .NET**

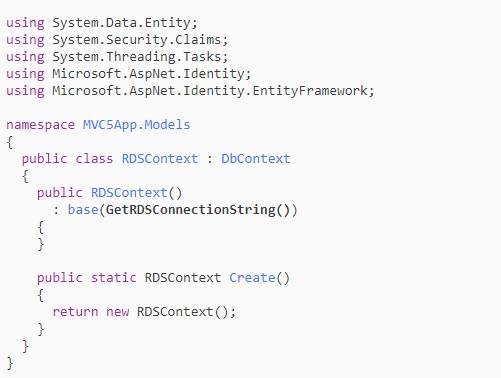
* **SQL Server** – Microsoft.EntityFrameworkCore.SqlServer
* **MySQL** – Pomelo.EntityFrameworkCore.MySql
* **PostgreSQL** – Npgsql.EntityFrameworkCore.PostgreSQL

## Connecting to a Database

Elastic Beanstalk provides connection information for attached DB instances in environment properties. Use ConfigurationManager.AppSettings to read the properties and configure a database connection.



Use the connection string to initialize your database context.

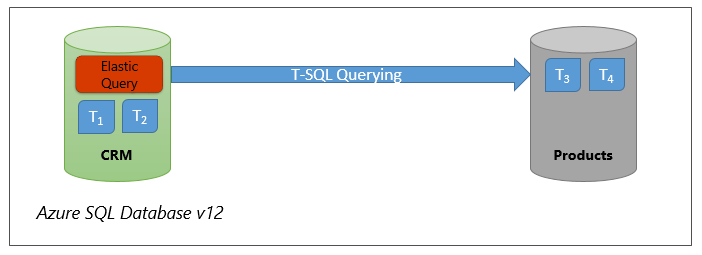


**Azure Web Service**

Cloud Computing is when you access computing services – like server, storage, networking, software – over the internet from a provider like Microsoft Azure, Amazon Web Service, Google Cloud Platform. Cloud Computing platforms, like Azure tend to be less expensive and more secure, reliable and flexible.

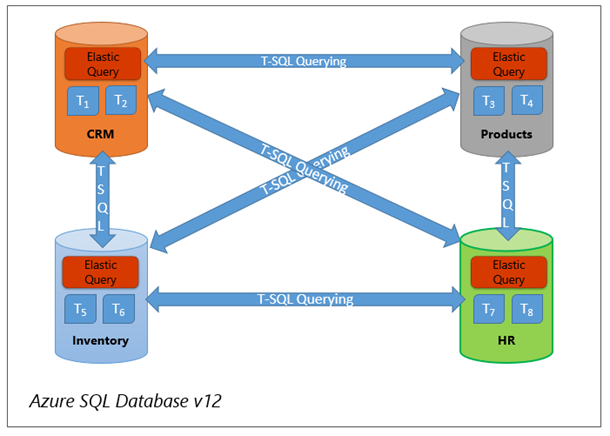
**Microsoft Azure** is a [cloud computing](https://en.wikipedia.org/wiki/Cloud_computing) service created by [Microsoft](https://en.wikipedia.org/wiki/Microsoft) for building, testing, deploying, and managing applications and services through Microsoft-managed [data centers](https://en.wikipedia.org/wiki/Data_center). It provides [software as a service (SaaS)](https://en.wikipedia.org/wiki/Software_as_a_service), [platform as a service (PaaS)](https://en.wikipedia.org/wiki/Platform_as_a_service) and [infrastructure as a service (IaaS)](https://en.wikipedia.org/wiki/Infrastructure_as_a_service) and supports many different [programming languages](https://en.wikipedia.org/wiki/Programming_language), tools and frameworks, including both Microsoft-specific and third-party software and systems.

Most notably, elastic database query now supports querying across databases in Azure SQL Database. This makes possible common cross-database querying tasks like selecting from a remote table into a local table.



Elastic database query now provides access to tables in remote Azure SQL Databases through a simple extension in the DDL for external data sources and external tables. You can define an external data source that, for instance, provides access to a remote database which stores reference data shared among all databases of your data tier. You can also easily copy the contents of tables from a remote database to another using INSERT INTO... SELECT statement.

It also allows for richer remote database querying topologies like the one illustrated in the following figure where a number of databases need access to each other’s tables.



The extensible APIs that you can use include .NET, .NET Core, Node.js, Java, Python, and MongoDB. If you’re using .NET, you can use DocumentDB API, MongoDB API, Graph API, or Table API. Java and Node.js will only work with the Document DB APIs, MongoDB APIs, and Graph APIs.

**Create Database Service For Node.js**

## **Open Azure Cloud Shell**

Azure Cloud Shell is a free, interactive shell that you can use to run the steps in this article. Common Azure tools are preinstalled and configured in Cloud Shell for you to use with your account. Select **Copy** to copy the code, paste it in Cloud Shell, and then press Enter to run it.

Select **Try It** in the upper-right corner of a code block.

Open Cloud Shell in your browser.

Select the **Cloud Shell** button on the menu in the upper-right corner of the [Azure portal](https://portal.azure.com/).

## **Create a resource group**

A [resource group](https://docs.microsoft.com/en-us/azure/azure-resource-manager/resource-group-overview#terminology) is a logical container into which Azure resources like web apps, databases, and storage accounts are deployed and managed. For example, you can choose to delete the entire resource group in one simple step later.

In the Cloud Shell, create a resource group with the [az group create](https://docs.microsoft.com/en-us/cli/azure/group?view=azure-cli-latest" \l "az-group-create) command. The following example creates a resource group named myResourceGroup in the South Central US location. To see all supported locations for App Service in **Free** tier, run the [az appservice list-locations --sku FREE](https://docs.microsoft.com/en-us/cli/azure/appservice?view=azure-cli-latest" \l "az-appservice-list-locations)command.

Azure CLI

az group create --name myResourceGroup --location "South Central US"

## **Create an Azure App Service plan**

In the Cloud Shell, create an App Service plan with the [az appservice plan create](https://docs.microsoft.com/en-us/cli/azure/appservice/plan?view=azure-cli-latest" \l "az-appservice-plan-create) command.

The following example creates an App Service plan named myAppServicePlan in the **Free** pricing tier:

Azure CLI

az appservice plan create --name myAppServicePlan --resource-group myResourceGroup --sku FREE

When the App Service plan has been created, the Azure CLI shows information similar to the following example:

JSON:

{

"adminSiteName": null,

"appServicePlanName": "myAppServicePlan",

"geoRegion": "South Central US",

"hostingEnvironmentProfile": null,

"id": "/subscriptions/0000-0000/resourceGroups/myResourceGroup/providers/Microsoft.Web/serverfarms/myAppServicePlan",

"kind": "app",

"location": "South Central US",

"maximumNumberOfWorkers": 1,

"name": "myAppServicePlan",

< JSON data removed for brevity. >

"targetWorkerSizeId": 0,

"type": "Microsoft.Web/serverfarms",

"workerTierName": null

}

In the Cloud Shell, create a web app in the myAppServicePlan App Service plan with the [az webapp create](https://docs.microsoft.com/en-us/cli/azure/webapp?view=azure-cli-latest#az-webapp-create) command.

# Bash and Powershell

az webapp create --resource-group myResourceGroup --plan myAppServicePlan --name <app\_name>

When the web app has been created, the Azure CLI shows output similar to the following example:

# Bash and Powershell

az webapp create --resource-group myResourceGroup --plan myAppServicePlan --name <app\_name>

JSON:

{

"availabilityState": "Normal",

"clientAffinityEnabled": true,

"clientCertEnabled": false,

"cloningInfo": null,

"containerSize": 0,

"dailyMemoryTimeQuota": 0,

"defaultHostName": "<app\_name>.azurewebsites.net",

"enabled": true,

< JSON data removed for brevity. >

}

### **Set Node.js runtime**

# Bash and Powershell

az webapp config appsettings set --resource-group myResourceGroup --name <app\_name> --settings WEBSITE\_NODE\_DEFAULT\_VERSION=10.14.1

Browse to your newly created web app. Replace <app\_name> with a unique app name.

http://<app\_name>.azurewebsites.net