Zero to Cloud-Native with IBM Cloud

Part 5: IBM Cloud Databases Provisioning and Configuration

Kevin Collins
kevincollins@us.ibm.com
Technical Sales Leader
IBM Cloud Enterprise Containers – Americas

Kunal Malhotra
kunal.malhotra3@ibm.com
Cloud Platform Engineer
IBM Cloud MEA

1 - IBM Cloud Databases

1 – 1 Introduction

Developing cloud native applications vs traditional applications is very different from a data perspective. In a traditional monolith application, it is very common to have one large SQL based database. Cloud native architecture on the other hand encourages multiple purpose-built databases. A best practice while developing cloud native application is isolating the data where each microservice manages its own data.

While you grow your application architecture and have a large number of independent databases, following good 12-factor app design, and also have a wide variety of databases, it's a lot to manage. There are a number of database types you may want to leverage from Redis as a caching database, to a high-performance document database like Cloudant, a general purpose SQL database like PostgreSQL, or maybe a high performance key value store database like etcd. This is one of the great parts of moving to cloud-native in being able to select the best database type for your service. You are not stuck to just one database like we typically see with a monolith. But how are you going to learn how to support, scale and secure each database.
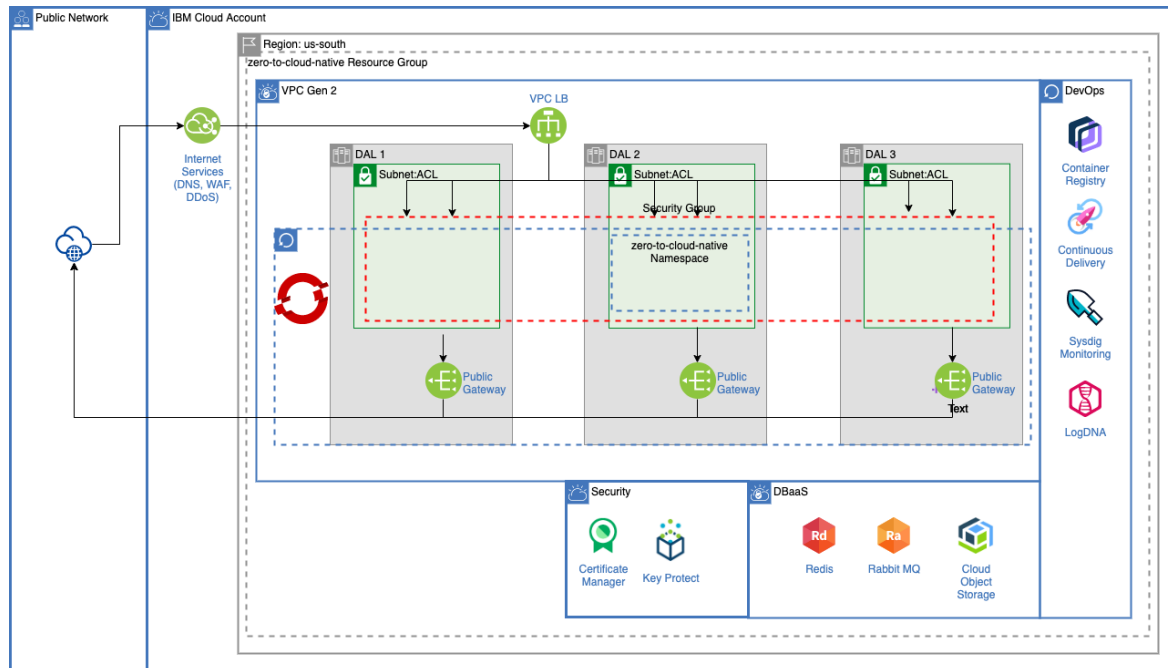
Enter IBM Cloud Databases. IBM Cloud Databases supports a wide range of databases that are managed for you.

This means that IBM takes care of all the hard parts of provisioning, supporting and maintaining different databases. IBM takes care of monitoring, backing up, making the database highly available, providing all the hardware and networking, and on top of that secures your databases. Another major benefit is these databases are provisioned in minutes. Providing all of these database services allows you to try out different databases to see which one best meets your application architecture. It's better to fail fast by developing say a quick MVP using MongoDB and being able to switch quickly to say a Cloudant database with minimal investment.

2 Tutorial – IBM Cloud Databases

Referring back to our application architecture of our tutorial application, we will be deploying two database / messaging services (Redis and RabbitMQ) that are part of IBM Cloud Databases.

You can read more about IBM Cloud Databases for Redis here:
https://cloud.ibm.com/docs/databases-for-redis?topic=databases-for-redis-getting-started

and Messages for RabbitMQ here:
https://cloud.ibm.com/docs/messages-for-rabbitmq?topic=messages-for-rabbitmq-getting-started

In the zero-to-cloud-native application we are going to develop and deploy, we are going to leverage caching to return data very quickly to the frontend. One of the most common use cases for using caching is to decrease data access latency. This is very important for frontend applications that may need to pull data to display in a user interface.

For this tutorial, we are going to simulate a long running task where we will replace the long running task with a cache using IBM Cloud Databases for Redis. We are going to do this through calling an API provided by IBM Cloud that retrieves a list of managed OpenShift versions. You can read and try the API here:
https://containers.cloud.ibm.com/global/swagger-global-api/#/beta/v2GetVersions

While this API does not take that long to run you will be able to understand how to implement caching for those long running APIs or database calls.

Before we can deploy our code, which as you will see leverages Redis, we need to deploy an instance that will store versions of IBM Cloud Managed OpenShift.

Before provisioning IBM Cloud Databases for Redis, we will need to authorize the service to access the Key Protect instance that you created in part 3. You will need to provide this authorization before provisioning the IBM Cloud Databases for Redis instance.

We will follow that same steps that we did in part 3, in authorizing Cloud Internet Services to access Certificate Manager.

1. Navigate to **IBM Cloud > Manage > Access (IAM) > Authorizations**.
2. Click **Create** and assign a source and target service. The source service is granted access to the target service based on the roles that you set in the next step.
- Source: IBM Cloud Databases for Redis
- Target: Key Protect
3. Specify a service instance for both the source and the target.
4. Assign the **Reader** role to allow IBM Cloud Databases for Redis to access the Key Protect. Then, click **Authorize**.

Grant a service authorization

First, select the source service to see which services it can be authorized to access. Then, select from the available target services, and assign all of the roles tha

Source service ⓘ

| Databases for Redis | × ∨ | in | Account | × ∨ |

Source service instance

| All instances | × ∨ |

Target service ⓘ

| Key Protect | × ∨ | in | Account | × ∨ |

Instance ID

| string equals | ∨ | Key Protect-zero-to-cloud-native (5b7fc5 | × ∨ | ⓘ |

Resource type

| string equals | ∨ | | ⓘ |

Resource ID

| string equals | ∨ | | ⓘ |

Authorize dependent services

Enabling the source service to delegate authorization to other dependent services automatically creates authorization policies for those services. Learn more.
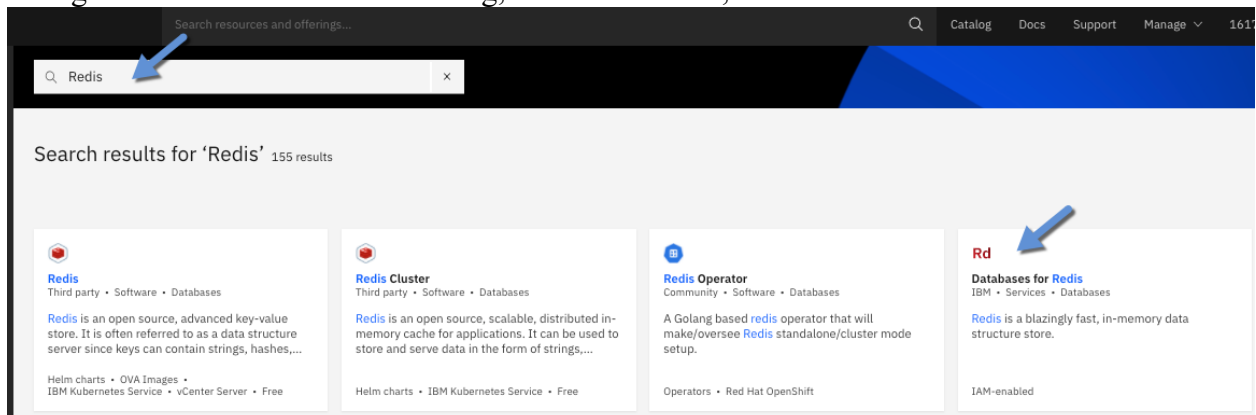
☐ Enable authorizations to be delegated

Service access ⓘ

☑ Reader 17 As a reader, you can perform read-only actions within a service such as viewing service-specific resources.

You will notice that I selected all instances in my account for Database for Redis to a specific instance of Key Protect. The reason for this is because I may want to create multiple Redis databases that I want to use Key Protect to encrypt the data, and don't want to have to go in and provide this authorization for each Redis database I create. If you do want to be more granular with your authorizations then you certain can and have that option.

Going back to the IBM Cloud Catalog, search for Redis, and click the Databases for Redis tile.

You will notice that there are several Redis options. If you want to manage, scale and secure Redis yourself you can install it directly in your cluster. For the reasons mentioned in the

introduction, we will be using the managed version of Redis named IBM Cloud Databases for Redis.

The first thing I'm going to select is Dallas for the region. I want the Redis instance to be in the same geographic area as my cluster for best performance.

Next, we need to give the Redis instance a name. Best practice is to have a Redis instance for each service that will leverage it. I will name it Databases for Redis-zero-to-cloud-native-versions which I can tell which application and service of the application that this instance of Redis will be used for.

I'll change the resource group to the zero-to-cloud-native resource group I've been using for all my IBM Cloud services.

The last thing I'm going to change is that I want the service to leverage the Key Protect instance that we created in part 3, to use the key I created and stored in Key Protect to encrypt data in the Redis database.

After entering all these settings, click create.

After clicking create, your database will be ready to use in a few minutes. This is why I strongly recommend using IBM Cloud Databases to manage and provision your cloud native databases, it doesn't get easier than this to setup and configure.

We will be using RabbitMQ as the messaging layer in-between microservices. Just like with Redis, I don't want to have to worry about how I'm going to provision, scale and secure RabbitMQ. I want IBM Cloud to do all the hard work for me, so I'm going to leverage IBM Cloud Messages for RabbitMQ managed service.

Just like we did the Redis, before provisioning IBM Cloud Messages for RabbitMQ, we will need to authorize the service to access our Key Protect instance that we created in part 3. You will need provide this authorization before provisioning the IBM Cloud Messages for RabbitMQ instance.

We will follow that same steps that we did above with Redis.

1. Navigate to **IBM Cloud > Manage > Access (IAM) > Authorizations**.
2. Click **Create** and assign a source and target service. The source service is granted access to the target service based on the roles that you set in the next step.
   - Source: Messages for RabbitMQ
   - Target: Key Protect

3. Specify a service instance for both the source and the target.
4. Assign the **Reader** role to allow IBM Cloud Messages for RabbitMQ to access the Key Protect.
5. Then, click **Authorize**.

This time, I'm going to give Messages for RabbitMQ authorization for every Key Protect service in the zero-to-cloud-native resource group so you can see different granularity options.

Now that we have authorized Messages for RabbitMQ with Key Protect we can provision our RabbitMQ service. To get started, go to the IBM Cloud Catalog, search for RabbitMQ, and then click on the Messages for RabbitMQ tile. Just like you saw this Redis, IBM Cloud offers different RabbitMQ services from do it yourself to a managed service with IBM Cloud Databases Messages for RabbitMQ.
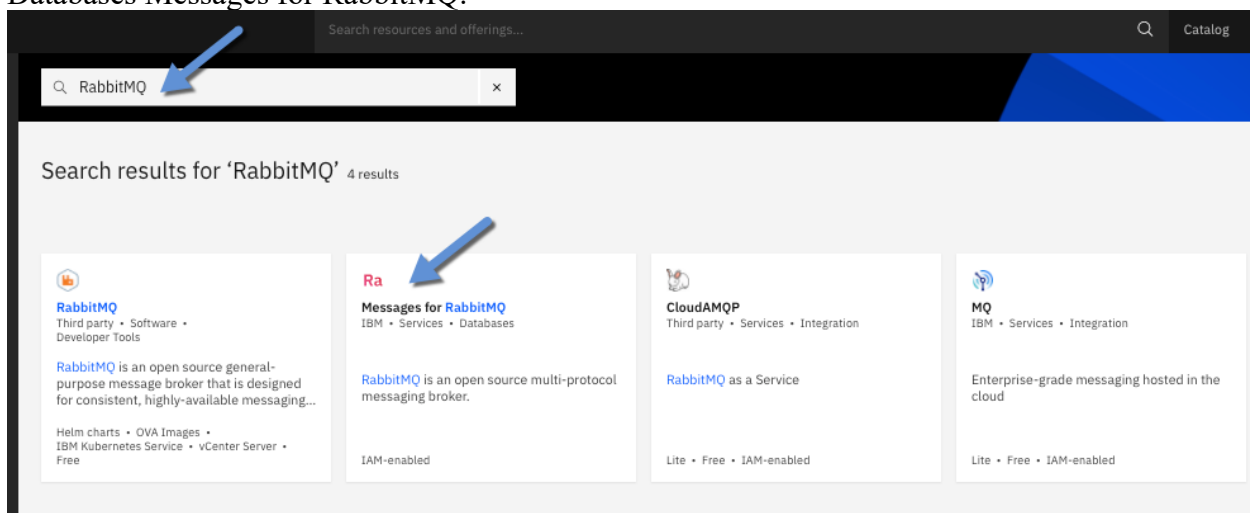
To create your Messages for Rabbit MQ instance you need to select a region. Like we have with all our other services we want to choose Dallas so all of our services are in the same geographic region.

Next, give you service a meaningful name such as Messages for RabbitMQ-zero-to-cloud-native. Select the zero-to-cloud-native resource group.
To leverage our Key Protect instance we create earlier, select the zero-to-cloud-native Key Protect instance along with the encryption key we created in step 3.

After entering all of these settings, click Create.

Just like the provisioning of Cloud Databases for Redis, that is all you have to do. Messages for RabbitMQ will be available in a couple of minutes.

Now that we have created all the various IBM Cloud services our application will be using, we can now start looking at the microservices code. However, before going through the code, \you need to have a development environment setup including a command line environment with various CLIs installed and configured, a development IDE, and other various tools that will make your cloud-native development life easier.