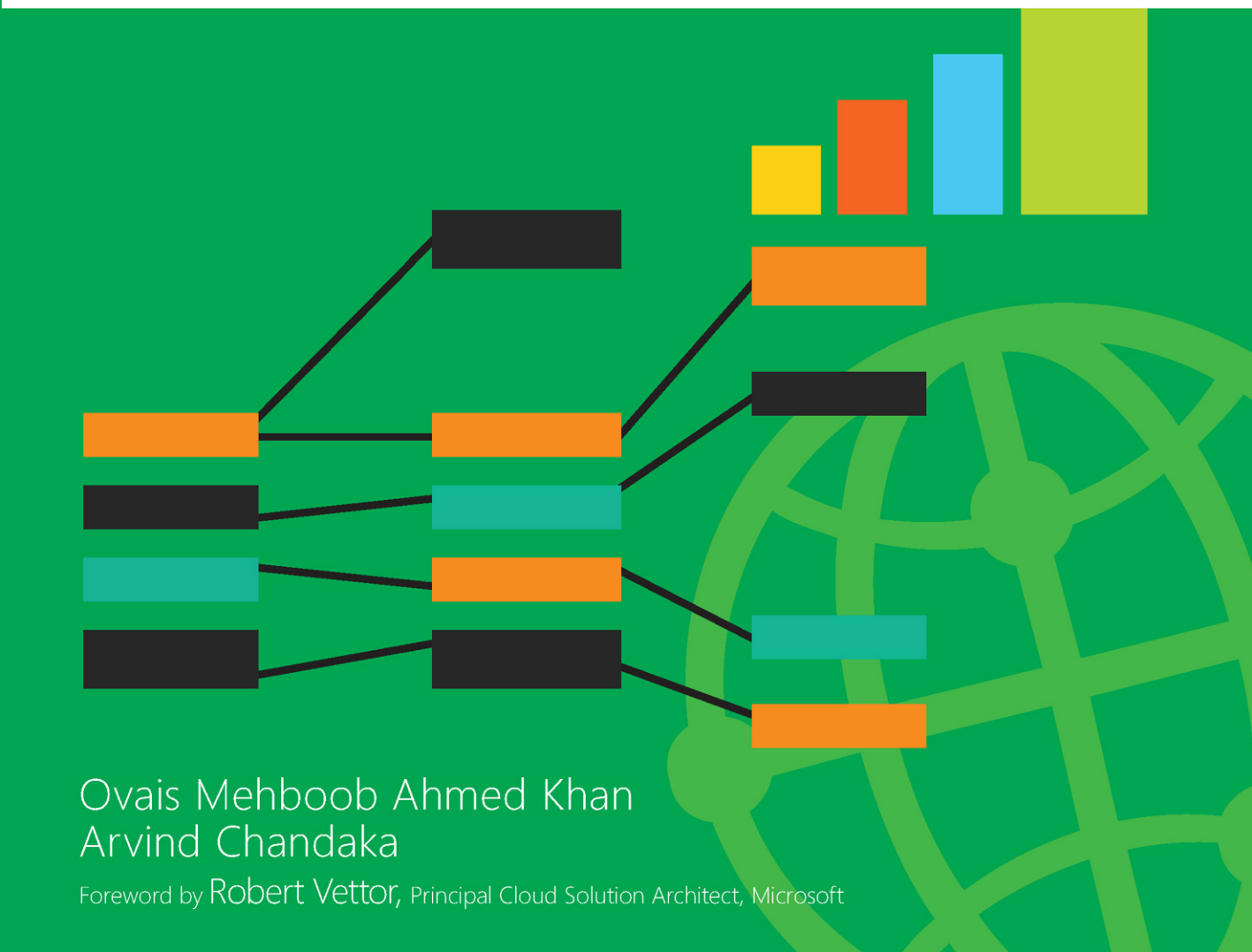
Microsoft

Developing Microservices Architecture on Microsoft Azure with Open Source Technologies



Ovais Mehboob Ahmed Khan
Arvind Chandaka

Foreword by Robert Vettor, Principal Cloud Solution Architect, Microsoft

Microsoft

Developing Microservices Architecture on Microsoft Azure with Open Source Technologies

Ovais Mehboob Ahmed Khan
Arvind Chandaka

Cosmos DB is a managed service that is globally distributed and supports multi-model databases. With Azure Cosmos DB, you can create both SQL and NoSQL databases. Azure Cosmos DB supports the following APIs.

- Azure Cosmos DB SQL API
- Azure Cosmos DB API for MongoDB
- Azure Cosmos DB Cassandra API
- Azure Cosmos DB Gremlin API
- Azure Cosmos DB Table API

We can choose any API on which to build the database, and you can use the same native SDK to connect to it. Azure Cosmos DB supports elastically scaling to various Azure regions worldwide. You can elastically scale throughput and storage and benefit from swift data access using any of the APIs mentioned above.

For the bid service, we will keep the bid information in a NoSQL database and thus, we used Azure Cosmos DB's MongoDB API. We chose the NoSQL database because of the high frequency of bids being made on active auctions and because NoSQL databases are optimized for insertions.

To provision Azure Cosmos DB in Azure, we use the same Terraform extension in VS Code. Make sure the Terraform extension is already installed in VS Code. (If you haven't done this yet, see the previous section.)

We will create a new Terraform file and name it `AzureCosmosDB.tf`. Listing 4-6 shows the script to provision Azure Cosmos DB for a Mongo API resource with Terraform:

LISTING 4-6 Provisioning an Azure Cosmos DB for a Mongo API resource with Terraform

```
resource "azurerm_cosmosdb_account" "cosmos-db" {
  name              =  "mongodboas"
  location          =  "westeurope"
  resource_group_name =  "OSS"
  offer_type         =  "Standard"
  kind               =  "MongoDB"
  consistency_policy {
    consistency_level       = "BoundedStaleness"
    max_interval_in_seconds = 10
    max_staleness_prefix    = 200
  }
  geo_location {
    location          = "westeurope"
    failover_priority = 0
  }
}
```

In Listing 4-6, the code is broken down like so:

- The `name` argument contains the actual name of the Cosmos DB resource you are provisioning, and the `location` argument is the Azure region where the resource will be provisioned.
- The `resource_group_name` argument states the name of the resource group, which is the same resource group we specified for MySQL server database.
- The `offer_type` argument denotes the plan. `Kind` should be set to MongoDB in order to create Azure Cosmos DB's API for MongoDB.
- The `consistent_policy` argument holds information specific to the consistency level and the max interval (in seconds) to represent the amount of staleness (in seconds) tolerated. The staleness prefix indicates the number of tolerated stale requests.
- Finally, the `geo_location` argument holds information specific to the location we need to initial provision this, and the `failover_priority` argument is set to `0`, which indicates a write region. The failover priority equals the total number of regions minus one.

To run the above script, you need to first run the `terraform init` command from VS Code, which opens the cloud shell in a VS Code terminal window. Next, you need to run `terraform plan` and then `terraform apply` as shown earlier in this chapter to create Azure Cosmos DB resource in Azure. See "Provision a MySQL database in Azure."

Database schema

The bid service database schema is shown in Table 4-3.

TABLE 4-3 Bid service database schema

Column Name	Type	Description
bidId	String	The unique ID of a bid
auctionId	String	An auction's ID
bidAmount	Number	The bid amount
userId	String	Bidder's user ID
bidDate	Date	Date when the bid was made

Create a bid service in the JavaSpring Boot framework

The bid service will be developed using the JavaSpring Boot framework, which is one of the most popular frameworks used for developing RESTful APIs in Java. You can easily create enterprise-grade APIs and accommodate all those scenarios that are essentials to be considered such as applying annotations and integrating with the Spring ecosystem that includes Spring JDBC, Spring ORM, and so on.

The JavaSpring Boot framework offers two flavors for creating APIs: Maven and Gradle. For the bid service, we will provision an application using Maven. First, create a new VS Code workspace/folder where the application will be created. VS Code provides an extension to provision the Spring Boot API.

Install the Java Extension Pack extension by going to the extension's palette from VS Code, as shown in Figure 4-10.

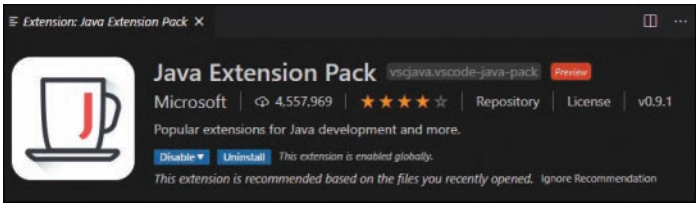


FIGURE 4-10 VS Code Java Extension Pack

Once the extension is installed, open the command palette by pressing Ctrl+Shift+P in VS Code, typing `spring`, as shown in Figure 4-11, and selecting the `Spring Initializr: Generate a Maven Project` option.

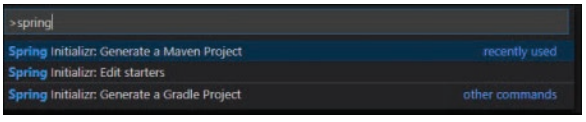


FIGURE 4-11 The above figure shows the way of creating JavaSpring Boot application using the Java Extension Pack installed in the previous step.

Once the **Maven** option is selected, it will take you through some wizard steps:

1. Under **Specify Project Language**, choose **Java**, as shown in Figure 4-12.

FIGURE 4-12 Select Java

2. For the I**nput Group Id For Your Project**, enter **com.onlineauctionweb,** as shown in Figure 4-13.

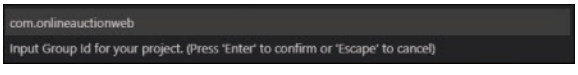


FIGURE 4-13 Group Id

3. For the **Input Artifact ID For Your Project**, enter **bidservice**, as shown in Figure 4-14.

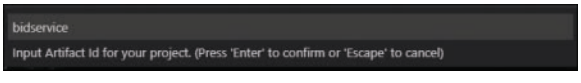


FIGURE 4-14 The Artifact ID is bidservice

4. Choose the JavaSpring Boot Application version. For our purposes, we will select **2.3.1**.

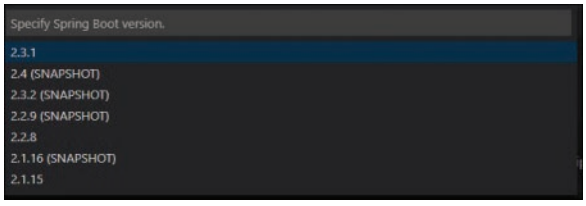


FIGURE 4-15 Choose the JavaSpring Boot application version

5. Select **Spring Boot DevTools** and **Spring Web** as dependencies, as shown in Figure 4-16.

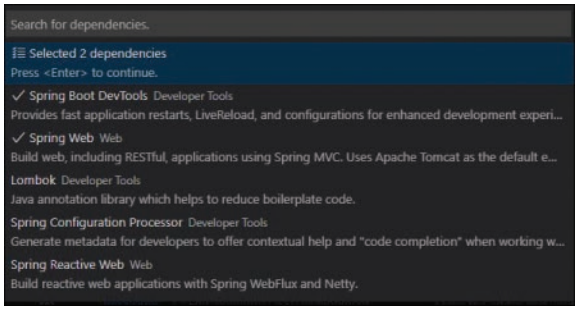


FIGURE 4-16 Select the dependencies used in the JavaSpring Boot Application

6. Choose the folder in which you will create the project files. You can choose the **bidservice** folder, where all the project files will be created.
7. Once the project is created, another instance of the VS Code is opened where the project is opened.

API methods in the bid service

The bid service is used to create bids. Table 4-4 shows the list of methods we will be adding in the bid service.

TABLE 4-4 Bid service methods

API Method	HTTP Verb	Signature	Description
Get Bid by Auction ID	GET	/bid	Returns bids based on Auction ID
Create Bid for Auction	POST	/bid	Creates a bid for the active auction

Create a bid model

Now that the basic project has been scaffolded, the first thing we need to do is create a bid model. This model contains some properties related to the bid table and is used for serializing/deserializing the JSON payload into the object.

Create a new folder named `Models` inside the `/bidservice` folder and add a new file named `BidDetail.java`. Listing 4-7 shows the code snippet of the `BidDetail` class:

LISTING 4-7 Bid detail entity

```
public class BidDetail{
    public String bidID;
    public String amount;
    public String customer;
    public String customerName;
    public String bidAt;
}
```

The `BidDetail` contains properties such as `bidID`, `amount`, `customer`, `customerName`, and `bidAt`.

Create a bid controller

Add a bid controller that will expose some methods to create and get bids from the Mongo DB database, and add a new class inside the `/bidservice` folder and name it `BidController`.

Get bids using the auctionID method

Add a method to return the bids by `auctionID`. Add a new method named `GetBidByAuctionID` that takes `auctionID` as a parameter and returns the list of auctions. Listing 4-8 shows the code snippet for the `GetBidByAuctionID` method.

LISTING 4-8 Get Bid by Auction ID method

```
@CrossOrigin(origins = "*", allowedHeaders = "*")
@RequestMapping(value = "/bid", method = RequestMethod.GET)
@ResponseBody
public ArrayList GetBidByAuctionID(String auctionID)
{
    ArrayList lst=new ArrayList();
     telemetryClient.trackTrace("Getting Bid by Auction ID");
    try {
        MongoClient mongoClient = new MongoClient(new MongoClientURI(
                "mongodb://mongodbtfx:xSaxMiCzTRAly2sb4H8IqpkidTmxHK36G3H8oIKb9q
                OfCQbjfSuMOHqC8pwRzpSLtOWbrfITjzsOe2FYDidWHw==@mongodbtfx.docu-
                ments.azure.com:10255/?ssl=true&replicaSet=globaldb"));
         DB db=  mongoClient.getDB("biddb");
         DBCollection coll=db.getCollection("bids");
         BasicDBObject doc = new BasicDBObject();
         doc.put("auctionId", auctionID);
         DBCursor cursor= coll.find(doc);
         Integer counter=1;
         while(cursor.hasNext()) {
```

```
                BasicDBObject obj = (BasicDBObject) cursor.next();
                BidDetail bidDetail=new BidDetail();
                bidDetail.bidID = counter.toString();
                bidDetail.amount = obj.getString("bidAmount");
                bidDetail.customer = obj.getString("userId");
                bidDetail.customerName= obj.getString("userName");
                bidDetail.bidAt=obj.getString("bidDate");
                lst.add(bidDetail);
                counter++;
            }
        } catch (UnknownHostException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    return lst;
}
```

In the method shown in Listing 4-8, we are first initializing the `ArrayList` instance and then initializing the `MongoClient` object and passing the connection string. The connection string can be obtained from the Azure Portal, by navigating to the Cosmos DB resource and opening the **Connection Strings** tab. You need to replace the connection string to connect with the Cosmos DB resource. Next, initialize the DB object and mention the exact name of DB, which in our case is `biddb`. The `DBCollection` is used to read the collection by specifying the collection name.

To find the bids based on Auction ID, use the `BasicDBObject` class and specify the key/value pair where the key refers to the column name in the bids collection and the value represents the parameter value passed in the method.

Also, you need to initialize the counter to start at 1 and then execute a loop to read all the bids for the auction. Finally, return the list of bids as a response. To use MongoDB API, you also need to modify the `pom.xml` file and add the MongoDB dependency as shown in Listing 4-9:

LISTING 4-9 Adding MongoDB library reference

```
<dependency>
      <groupId>org.mongodb</groupId>
      <artifactId>mongo-java-driver</artifactId>
      <version>2.12.4</version>
</dependency>
```

See the code repository mentioned in this book's Introduction for a complete list of dependencies needed to build the application.

Create the bid method

Add a new method to create bid and name it `CreateBid`. The method takes four parameters: `bidAmount`, `auctionID`, `userID`, and `userName`. Listing 4-10 shows the complete `CreateBid` method code snippet: