# Get started with cloud development using Java on Azure

- Article
- 09/01/2022

This article walks you through setting up a development environment for Azure development in Java. You'll then create some Azure resources and connect to them to do some basic tasks, like uploading a file or deploying a web application. When you're finished, you'll be ready to start using Azure services in your own Java applications.

## Prerequisites

- An Azure account. If you don't have one, get a free trial.
- Azure Cloud Shell or Azure CLI 2.0.
- Java 8, which is included in Azure Cloud Shell.
- Maven 3, which is included in Azure Cloud Shell.

## Set up authentication

Your Java application needs *read* and *create* permissions in your Azure subscription to run the sample code in this tutorial. Create a service principal, and configure your application to run with its credentials. Service principals provide a way to create a noninteractive account associated with your identity to which you grant only the privileges your app needs to run.

Create a service principal by using the Azure CLI 2.0, and capture the output:

Azure CLICopy

```
az ad sp create-for-rbac \
    --name AzureJavaTest \
    --role Contributor \
    --scopes /subscriptions/<your-subscription-ID>
```

This command gives you a reply in the following format:

JSONCopy

```
{
  "appId": "a487e0c1-82af-47d9-9a0b-af184eb87646d",
  "displayName": "AzureJavaTest",
  "name": "http://AzureJavaTest",
  "password": "aaaaaaaa-aaaa-aaaa-aaaa-aaaaaaaaaaaa",
  "tenant": "tttttttt-tttt-tttt-tttt-tttttttttttt"
}
```

Next, configure the environment variables:

- `AZURE_SUBSCRIPTION_ID`: Use the *id* value from `az account show` in the Azure CLI 2.0.
- `AZURE_CLIENT_ID`: Use the *appId* value from the output taken from a service principal output.
- `AZURE_CLIENT_SECRET`: Use the *password* value from the service principal output.
- `AZURE_TENANT_ID`: Use the *tenant* value from the service principal output.

For more authentication options, see the Azure Identity client library for Java.

## Tooling

### Create a new Maven project

**Note**

This article uses the Maven build tool to build and run the sample code. Other build tools, such as Gradle, also work with the Azure SDK for Java.

Create a Maven project from the command line in a new directory on your system.

BashCopy

```
mkdir java-azure-test
cd java-azure-test
mvn archetype:generate -DgroupId=com.fabrikam -DartifactId=AzureApp \
-DarchetypeArtifactId=maven-archetype-quickstart -DinteractiveMode=false
```

This step creates a basic Maven project under the *testAzureApp* directory. Add the following entries into the project's *pom.xml* file to import the libraries used in the sample code in this tutorial.

XMLCopy

```
<dependency>
  <groupId>com.azure</groupId>
```

```
  <artifactId>azure-identity</artifactId>
  <version>1.3.2</version>
</dependency>
<dependency>
  <groupId>com.azure.resourcemanager</groupId>
  <artifactId>azure-resourcemanager</artifactId>
  <version>2.6.0</version>
</dependency>
<dependency>
  <groupId>com.azure</groupId>
  <artifactId>azure-storage-blob</artifactId>
  <version>12.8.0</version>
</dependency>
<dependency>
  <groupId>com.microsoft.sqlserver</groupId>
  <artifactId>mssql-jdbc</artifactId>
  <version>6.2.1.jre8</version>
</dependency>
<dependency>
  <groupId>org.slf4j</groupId>
  <artifactId>slf4j-simple</artifactId>
  <version>1.7.33</version>
</dependency>
```

Add a `build` entry under the top-level `project` element to use the maven-exec-plugin to run the samples. maven-compiler-plugin is used to configure the source code and generated classes for Java 8.

XMLCopy

```
<build>
  <plugins>
    <plugin>
      <groupId>org.codehaus.mojo</groupId>
      <artifactId>exec-maven-plugin</artifactId>
      <version>3.0.0</version>
      <configuration>
        <mainClass>com.fabrikam.App</mainClass>
      </configuration>
    </plugin>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-compiler-plugin</artifactId>
      <version>3.8.1</version>
      <configuration>
        <source>1.8</source>
        <target>1.8</target>
      </configuration>
    </plugin>
  </plugins>
</build>
```

## Create a Linux virtual machine

Create a new file named *App.java* in the project's *src/main/java/com/fabrikam* directory, and paste in the following block of code. Update the `userName` and `sshKey` variables with real values for your machine. The code creates a new Linux virtual machine (VM) with the name `testLinuxVM` in the resource group `sampleResourceGroup` running in the US East Azure region.

JavaCopy

```
package com.fabrikam;

import com.azure.core.credential.TokenCredential;
import com.azure.core.http.policy.HttpLogDetailLevel;
import com.azure.core.management.AzureEnvironment;
import com.azure.core.management.Region;
import com.azure.core.management.profile.AzureProfile;
import com.azure.identity.AzureAuthorityHosts;
import com.azure.identity.DefaultAzureCredentialBuilder;
import com.azure.resourcemanager.AzureResourceManager;
import com.azure.resourcemanager.compute.models.KnownLinuxVirtualMachineImage;
import com.azure.resourcemanager.compute.models.VirtualMachine;
import com.azure.resourcemanager.compute.models.VirtualMachineSizeTypes;

public class App {

    public static void main(String[] args) {

        final String userName = "YOUR_VM_USERNAME";
        final String sshKey = "YOUR_PUBLIC_SSH_KEY";

        try {
            TokenCredential credential = new DefaultAzureCredentialBuilder()
                    .authorityHost(AzureAuthorityHosts.AZURE_PUBLIC_CLOUD)
                    .build();

            // If you don't set the tenant ID and subscription ID via environment
variables,
            // change to create the Azure profile with tenantId, subscriptionId, and
Azure environment.
            AzureProfile profile = new AzureProfile(AzureEnvironment.AZURE);

            AzureResourceManager azureResourceManager =
AzureResourceManager.configure()
                    .withLogLevel(HttpLogDetailLevel.BASIC)
                    .authenticate(credential, profile)
                    .withDefaultSubscription();

            // Create an Ubuntu virtual machine in a new resource group.
            VirtualMachine linuxVM =
azureResourceManager.virtualMachines().define("testLinuxVM")
                    .withRegion(Region.US_EAST)
```

```
                    .withNewResourceGroup("sampleVmResourceGroup")
                    .withNewPrimaryNetwork("10.0.0.0/24")
                    .withPrimaryPrivateIPAddressDynamic()
                    .withoutPrimaryPublicIPAddress()

.withPopularLinuxImage(KnownLinuxVirtualMachineImage.UBUNTU_SERVER_18_04_LTS)
                    .withRootUsername(userName)
                    .withSsh(sshKey)
                    .withSize(VirtualMachineSizeTypes.STANDARD_D3_V2)
                    .create();

        } catch (Exception e) {
            System.out.println(e.getMessage());
            e.printStackTrace();
        }
    }
}
```

Run the sample from the command line.

BashCopy

```
mvn compile exec:java
```

You'll see some REST requests and responses in the console as the SDK makes the underlying calls to the Azure REST API to configure the VM and its resources. After the program finishes, verify the VM in your subscription with the Azure CLI 2.0.

Azure CLICopy

```
az vm list --resource-group sampleVmResourceGroup
```

After you've verified that the code worked, use the CLI to delete the VM and its resources.

Azure CLICopy

```
az group delete --name sampleVmResourceGroup
```

## Deploy a web app from a GitHub repo

Replace the main method in *App.java* with the following one. Update the `appName` variable to a unique value before you run the code. This code deploys a web application from the `master` branch in a public GitHub repo into a new Azure App Service Web App running in the free pricing tier.

JavaCopy

```
    public static void main(String[] args) {
        try {

            final String appName = "YOUR_APP_NAME";

            TokenCredential credential = new DefaultAzureCredentialBuilder()
                    .authorityHost(AzureAuthorityHosts.AZURE_PUBLIC_CLOUD)
                    .build();

            // If you don't set the tenant ID and subscription ID via environment
variables,
            // change to create the Azure profile with tenantId, subscriptionId, and
Azure environment.
            AzureProfile profile = new AzureProfile(AzureEnvironment.AZURE);

            AzureResourceManager azureResourceManager =
AzureResourceManager.configure()
                    .withLogLevel(HttpLogDetailLevel.BASIC)
                    .authenticate(credential, profile)
                    .withDefaultSubscription();

            WebApp app = azureResourceManager.webApps().define(appName)
                    .withRegion(Region.US_WEST2)
                    .withNewResourceGroup("sampleWebResourceGroup")
                    .withNewWindowsPlan(PricingTier.FREE_F1)
                    .defineSourceControl()
                    .withPublicGitRepository(
                            "https://github.com/Azure-Samples/app-service-web-java-
get-started")
                    .withBranch("master")
                    .attach()
                    .create();

        } catch (Exception e) {
            System.out.println(e.getMessage());
            e.printStackTrace();
        }
    }
```

Run the code as before using Maven.

BashCopy

```
mvn clean compile exec:java
```

Open a browser pointed to the application by using the CLI.

Azure CLICopy

```
az webapp browse --resource-group sampleWebResourceGroup --name YOUR_APP_NAME
```

Remove the web app and plan from your subscription after you've verified the deployment.

Azure CLICopy

```
az group delete --name sampleWebResourceGroup
```

## Connect to an Azure SQL database

Replace the current main method in *App.java* with the following code. Set real values for the variables. This code creates a new SQL database with a firewall rule that allows remote access. Then the code connects to it by using the SQL Database JBDC driver.

JavaCopy

```
    public static void main(String args[]) {
        // Create the db using the management libraries.
        try {
            TokenCredential credential = new DefaultAzureCredentialBuilder()
                    .authorityHost(AzureAuthorityHosts.AZURE_PUBLIC_CLOUD)
                    .build();

            // If you don't set the tenant ID and subscription ID via environment
variables,
            // change to create the Azure profile with tenantId, subscriptionId, and
Azure environment.
            AzureProfile profile = new AzureProfile(AzureEnvironment.AZURE);

            AzureResourceManager azureResourceManager =
AzureResourceManager.configure()
                    .withLogLevel(HttpLogDetailLevel.BASIC)
                    .authenticate(credential, profile)
                    .withDefaultSubscription();

            final String adminUser = "YOUR_USERNAME_HERE";
            final String sqlServerName = "YOUR_SERVER_NAME_HERE";
            final String sqlDbName = "YOUR_DB_NAME_HERE";
            final String dbPassword = "YOUR_PASSWORD_HERE";
            final String firewallRuleName = "YOUR_RULE_NAME_HERE";

            SqlServer sampleSQLServer =
azureResourceManager.sqlServers().define(sqlServerName)
                    .withRegion(Region.US_EAST)
                    .withNewResourceGroup("sampleSqlResourceGroup")
                    .withAdministratorLogin(adminUser)
                    .withAdministratorPassword(dbPassword)
                    .defineFirewallRule(firewallRuleName)
                        .withIpAddressRange("0.0.0.0","255.255.255.255")
                        .attach()
                    .create();
```

```
            SqlDatabase sampleSQLDb =
sampleSQLServer.databases().define(sqlDbName).create();

            // Assemble the connection string to the database.
            final String domain = sampleSQLServer.fullyQualifiedDomainName();
            String url = "jdbc:sqlserver://"+ domain + ":1433;" +
                    "database=" + sqlDbName +";" +
                    "user=" + adminUser+ "@" + sqlServerName + ";" +
                    "password=" + dbPassword + ";" +

"encrypt=true;trustServerCertificate=false;hostNameInCertificate=*.database.windows.n
et;loginTimeout=30;";

            // Connect to the database, create a table, and insert an entry into it.
            try (Connection conn = DriverManager.getConnection(url)) {
                String createTable = "CREATE TABLE CLOUD (name varchar(255), code
int);";
                String insertValues = "INSERT INTO CLOUD (name, code) VALUES
('Azure', 1);";
                String selectValues = "SELECT * FROM CLOUD";
                try (Statement createStatement = conn.createStatement()) {
                    createStatement.execute(createTable);
                }
                try (Statement insertStatement = conn.createStatement()) {
                    insertStatement.execute(insertValues);
                }
                try (Statement selectStatement = conn.createStatement();
                     ResultSet rst = selectStatement.executeQuery(selectValues)) {
                    while (rst.next()) {
                        System.out.println(rst.getString(1) + " " +
rst.getString(2));
                    }
                }
            }
        } catch (Exception e) {
            System.out.println(e.getMessage());
            System.out.println(e.getStackTrace().toString());
        }
    }
```

Run the sample from the command line.

BashCopy

```
mvn clean compile exec:java
```

Then clean up the resources by using the CLI.

Azure CLICopy

```
az group delete --name sampleSqlResourceGroup
```

## Write a blob into a new storage account

Replace the current main method in *App.java* with the following code. This code creates an Azure storage account. Then the code uses the Azure Storage libraries for Java to create a new text file in the cloud.

JavaCopy

```
    public static void main(String[] args) {

        try {
            TokenCredential tokenCredential = new DefaultAzureCredentialBuilder()
                    .authorityHost(AzureAuthorityHosts.AZURE_PUBLIC_CLOUD)
                    .build();

            // If you don't set the tenant ID and subscription ID via environment
variables,
            // change to create the Azure profile with tenantId, subscriptionId, and
Azure environment.
            AzureProfile profile = new AzureProfile(AzureEnvironment.AZURE);

            AzureResourceManager azureResourceManager =
AzureResourceManager.configure()
                    .withLogLevel(HttpLogDetailLevel.BASIC)
                    .authenticate(tokenCredential, profile)
                    .withDefaultSubscription();

            // Create a new storage account.
            String storageAccountName = "YOUR_STORAGE_ACCOUNT_NAME_HERE";
            StorageAccount storage =
azureResourceManager.storageAccounts().define(storageAccountName)
                    .withRegion(Region.US_WEST2)
                    .withNewResourceGroup("sampleStorageResourceGroup")
                    .create();

            // Create a storage container to hold the file.
            List<StorageAccountKey> keys = storage.getKeys();
            PublicEndpoints endpoints = storage.endPoints();
            String accountName = storage.name();
            String accountKey = keys.get(0).value();
            String endpoint = endpoints.primary().blob();

            StorageSharedKeyCredential credential = new
StorageSharedKeyCredential(accountName, accountKey);

            BlobServiceClient storageClient = new BlobServiceClientBuilder()
                    .endpoint(endpoint)
                    .credential(credential)
                    .buildClient();

            // Container name must be lowercase.
            BlobContainerClient blobContainerClient =
storageClient.getBlobContainerClient("helloazure");
```

```
            blobContainerClient.create();

            // Make the container public.
            blobContainerClient.setAccessPolicy(PublicAccessType.CONTAINER, null);

            // Write a blob to the container.
            String fileName = "helloazure.txt";
            String textNew = "Hello Azure";

            BlobClient blobClient = blobContainerClient.getBlobClient(fileName);
            InputStream is = new ByteArrayInputStream(textNew.getBytes());
            blobClient.upload(is, textNew.length());

        } catch (Exception e) {
            System.out.println(e.getMessage());
            e.printStackTrace();
        }
    }
```

Run the sample from the command line.

BashCopy

```
mvn clean compile exec:java
```

You can browse for the *helloazure.txt* file in your storage account through the Azure portal or with Azure Storage Explorer.

Clean up the storage account by using the CLI.

Azure CLICopy

```
az group delete --name sampleStorageResourceGroup
```

## Explore more samples

To learn more about how to use the Azure management libraries for Java to manage resources and automate tasks, see our sample code for virtual machines, web apps, and SQL database.

## Reference and release notes

A reference is available for all packages.

## Get help and give feedback

Post questions to the community on Stack Overflow. Report bugs and open issues against the Azure SDK for Java in the GitHub repository.