**Migration of services from AWS to Azure**

* Migrating services from AWS to Azure involves several steps to ensure a successful transition.
* List out the services and resources currently running in AWS. Identify the dependencies, configurations, and data associated with each service.
* Assess the equivalent services and features available in Azure that will replace the AWS services.

**High-level panning:**

## **Pre-Migration Planning:**

1. Assessment of the current AWS environment
2. Identifying services to migrate
3. Analyzing dependencies and interconnections
4. Determining the migration approach (lift-and-shift, re-platform, or re-architect)
5. Setting migration goals and timelines
6. Creating a migration team and assigning roles

## **Azure Account Setup:**

1. Creating an Azure account
2. Configuring subscriptions and resource groups
3. Setting up networking and security policies
4. Configuring Azure Active Directory (AAD) and permissions

## **AWS to Azure Mapping:**

1. Mapping AWS services to their Azure equivalents
2. Understanding the similarities and differences between AWS and Azure offerings
3. Translating AWS-specific configurations to Azure equivalents

## **Application Migration:**

1. Identifying application dependencies
2. Replicating application infrastructure in Azure
3. Migrating virtual machines, containers, and serverless functions
4. Configuring and optimizing application performance in Azure

## **Networking and Security:**

1. Replicating network architecture in Azure (Virtual Networks, Subnets, Load Balancers, etc.)
2. Configuring Azure network security groups and access control
3. Setting up Azure VPN or ExpressRoute for hybrid connectivity
4. Implementing Azure security best practices and compliance requirements
5. **Storage and Database Migration:**
6. Migrating AWS storage services (S3, EBS, Glacier) to Azure Blob Storage
7. Replicating AWS database services (RDS, DynamoDB, Redshift) to Azure counterparts (Azure SQL Database, Cosmos DB, Azure Synapse Analytics, etc.)
8. Data replication, synchronization, and cutover strategies.

## **Testing and Validation**

1. Defining a comprehensive testing strategy
2. Conducting performance and load testing
3. Ensuring compatibility and functionality across different environments

**AWS IAM to Azure AD:**

## **Using the AWS Single Sign-On (SSO) integration with Azure AD**

The AWS SSO integration with Azure AD allows you to automatically synchronize users and groups between AWS IAM and Azure AD. This means that you can create users and groups in Azure AD, and they will automatically be created in AWS IAM, and vice versa.

To use the AWS SSO integration with Azure AD, you need to:

1. Create an Azure AD tenant.
2. Enable the AWS SSO integration with Azure AD in your Azure AD tenant.
3. Create a SAML connection between your Azure AD tenant and AWS.
4. Configure your AWS IAM users and groups to use the SAML connection.
5. Once you have configured the AWS SSO integration with Azure AD, users and groups will be automatically synchronized between AWS IAM and Azure AD.

Here are the steps on how to enable the AWS SSO integration with Azure AD in your Azure AD tenant:

1. Go to the Azure AD portal.
2. In the left navigation pane, click Azure Active Directory, then click Enterprise Applications.
3. Click New application.
4. In the Add from the gallery section, type AWS IAM Identity Center in the search box.
5. Click AWS IAM Identity Center.
6. Click Create.

* In the Application properties section, enter the following information:
* Application ID: The application ID for your AWS IAM Identity Center instance.
* Sign-on URL: The sign-on URL for your AWS IAM Identity Center instance.
* Application Display Name: The display name for your AWS IAM Identity Center instance.

1. Click create

Once you have created the application, you need to configure your AWS IAM users and groups to use the SAML connection.

Here are the steps on how to configure your AWS IAM users and groups to use the SAML connection:

1. Go to the AWS IAM console.
2. In the left navigation pane, click Users.
3. Select the users that you want to configure to use the SAML connection.
4. Click Actions, then click Edit.
5. In the Edit user dialog box, select the SAML tab.
6. Enter the following information:

* SAML Identity Provider (IdP) ARN: The ARN for your AWS IAM Identity Center instance.
* SAML Signing Certificate: The signing certificate for your AWS IAM Identity Center instance.

1. Click Save.

Once you have configured your AWS IAM users and groups to use the SAML connection, they will automatically be synchronized with Azure AD.

In order to migrate our services from AWS to Azure first we need to start with the network and migrate from AWS to Azure.

**AWS VPC to Azure Virtual network:**

## **Step 1: Create an Azure VNET**

The first step is to create an Azure VNET. This is the equivalent of a VPC in AWS. To create an Azure VNET, you will need to specify the following:

1. The name of the VNET.
2. The location of the VNET.
3. The size of the VNET.
4. The subnet configuration.

**Procedure:**

* In the search box at the top of the portal, enter Virtual networks. Select Virtual networks in the search results.
* Select + Create.
* In the Basics tab of Create virtual network, enter the required details like subscription, Resource group, Name of the virtual network, and Region.
* Select the IP Addresses tab or Next: IP Addresses >, and enter the IP address range.

## **Step 2: Create subnets**

Once you have created an Azure VNET, you will need to create subnets. Subnets are the equivalent of subnets in AWS. To create a subnet, you will need to specify the following:

1. The name of the subnet.
2. The address range of the subnet.
3. The network security group for the subnet.

## **Step 3: Create an Internet gateway**

An internet gateway is the equivalent of an internet gateway in AWS. It allows your Azure VNET to communicate with the public internet. To create an internet gateway, you will need to specify the following:

1. The name of the internet gateway
2. The location of the internet gateway

## **Step 4: Create a NAT gateway**

A NAT gateway is the equivalent of a NAT gateway in AWS. It allows your Azure VNET to communicate with the internet, but it hides your private IP addresses. To create a NAT gateway, you will need to specify the following:

1. The name of the NAT gateway
2. The location of the NAT gateway

## **Step 5: Create security groups**

Security groups are the equivalent of security groups in AWS. They allow you to control the traffic that is allowed to enter and exit your Azure VNET. To create a security group, you will need to specify the following:

1. The name of the security group
2. The rules for the security group

## **Step 6: Create peering connections**

Peering connections are the equivalent of VPC peering connections in AWS. They allow you to connect two Azure VNETs together. To create a peering connection, you will need to specify the following:

1. The name of the peering connection
2. The Azure VNET that you want to peer with

**AWS EC2 to Azure Virtual Network:**

# **Azure Migrate**

## **migrate windows servers from aws to azure**

create project:

1. add subscription and resource group

2. enter project details(migrate project and geography) and create

after creating we can see server assessment and server migration tools are added in our project

1. discover servers from server migration

* In Discover select virtualization type(physical or other)
* Select target region
* Create resource
* Once it is done we need to download the assessment and migration tool which we need to install on the appliance server so that we can start discovering things from AWS to Azure
* After deployment we need to choose appliance type(1. install replication appliance 2. install a scale-out appliance)
* After selecting the replication appliance(since we are doing this migration for the first time) we will get two software which are supposed to download
* The first one is appliance software and the second one is the registration key these two pieces of software help us in installing the appliance in aws instance.
* copy the two software in the appliance server.
* Install the software in the appliance server.
* Go ahead with the configuration of the server. while configuring we are asked for the key that we downloaded earlier and we need to add it.
* After installation is completed it will ask for us to save the passphrase
* We are supposed to add user accounts here for our servers
* After configuring we need to reboot the machine

**Source files:** /c/programData/ASR/home/svsystems/pushinstallsvc/repository

based on the operating system we need to copy this on the target machine and we need to install it.

* Now go back to Azure portal and see whether it is reflected in the portal
* Click on Discover again so that you will find the configuration server
* Finalize registration on it.
* Now check the replicate option to start replicating the discovered machines.
* Select replication appliance and process server and click next
* Select the virtual machine that migrated
* Now select the resource group, network, and subnet, and click next
* Select VM size and choose disc and click on replicate the application.
* After replication is done we can migrate that machine
* Select the machine and click on migrate to migrate the same machine
* Now the machine is migrated

## **linux server migrating from aws to azure:**

1. Login to your Linux machine
2. Copy the software from the appliance server
3. In the appliance server go to /c/programData/ASR/home/svsystems/pushinstallsvc/repository and copy the software to Linux machine
4. Access to the root account
5. In /etc/ssh/sshd\_config file enable permitRootLogin and PasswordAuthentication and restart the service
6. In the appliance server, there is a file called cspconfig. open that file and add Linux account details
7. Follow the steps

* mkdir /tmp/MobSvcInstaller
* tar -C /tmp/MobSvcInstaller -xvf <Installer tarball>
* cd /tmp/MobSvcInstaller

sudo ./install -r MS -q

1. Register this machine with the appliance server

/usr/local/ASR/Vx/bin/unifiedagentconf.sh and give the appliance machine ip and passphrase

1. After this step, your machine will be shown in Azure portal
2. Now you can discover this and replicate and migrate**.**

**AWS S3 to Azure Blob**

AzCopy v10 (Preview) now supports Amazon Web Services (AWS) S3 as a data source. You can now copy an entire AWS S3 bucket, or even multiple buckets, to Azure Blob Storage using AzCopy.

AzCopy copies data from AWS S3 with high throughput by scaling out copy jobs to multiple Azure Storage servers. AzCopy relies on the new Azure Storage REST API operation Put Block from URL, which copies data directly from a given URL. Using Put Block from URL, AzCopy v10 moves data from an AWS S3 bucket to an Azure Storage account, without first copying the data to the client machine where AzCopy is running. Instead, Azure Storage performs the copy operation directly from the source.

## **Configure access and authorize AzCopy with Azure and AWS:**

First, you will need to install AzCopy on your machine. After that, you will need to approve AzCopy with Microsoft Azure and AWS. To authorize AWS S3, you have to use an AWS access key and a secret access key.

### **Copy an AWS S3 object to Azure blob:**

**syntax:**

azcopy copy 'https://s3.amazonaws.com/<bucket-name>/<object-name>' 'https://<storage-account-name>.blob.core.windows.net/<container-name>/<blob-name>'

**Example:**

azcopy copy 'https://s3.amazonaws.com/mybucket/myobject' 'https://mystorageaccount.blob.core.windows.net/mycontainer/myblob'

### **Copy and migrate Amazon S3 directory to Azure:**

**Syntax:**

azcopy copy 'https://s3.amazonaws.com/<bucket-name>/<directory-name>' 'https://<storage-account-name>.blob.core.windows.net/<container-name>/<directory-name>' --recursive=true

**Example:**

azcopy copy 'https://s3.amazonaws.com/mybucket/mydirectory' 'https://mystorageaccount.blob.core.windows.net/mycontainer/mydirectory' --recursive=true

### **Copy the contents of a directory**

**Syntax:**

### azcopy copy 'https://s3.amazonaws.com/<bucket-name>/<directory-name>/\*' 'https://<storage-account-name>.blob.core.windows.net/<container-name>/<directory-name>' --recursive=true

**Example:**

azcopy copy 'https://s3.amazonaws.com/mybucket/mydirectory/\*' 'https://mystorageaccount.blob.core.windows.net/mycontainer/mydirectory' --recursive=true

### **Copy an Amazon S3 bucket to Azure blob storage:**

**Syntax:**

azcopy copy 'https://s3.amazonaws.com/<bucket-name>' 'https://<storage-account-name>.blob.core.windows.net/<container-name>' --recursive=true

**Example:**

azcopy copy 'https://s3.amazonaws.com/mybucket' 'https://mystorageaccount.blob.core.windows.net/mycontainer' --recursive=true

### **Copy all buckets in a specific S3 region:**

**Syntax:**

azcopy copy 'https://s3-<region-name>.amazonaws.com/' 'https://<storage-account-name>.blob.core.windows.net' --recursive=true

**Example:**

azcopy copy 'https://s3-rds.eu-north-1.amazonaws.com' 'https://mystorageaccount.blob.core.windows.net' --recursive=true

### **Copy all buckets in all regions:**

**Syntax:**

azcopy copy 'https://s3.amazonaws.com/' 'https://<storage-account-name>.blob.core.windows.net' --recursive=true

**Example:**

azcopy copy 'https://s3.amazonaws.com' 'https://mystorageaccount.blob.core.windows.net' --recursive=true

## **Hosting S3 website in azure bob:**

### **Create a storage account to host the static website:**

1. Click + Add to create a new resource.
2. Search for and select Storage Account.
3. For Resource Group, select the existing resource group if it is not already selected.
4. Enter a Storage account name that begins with staticsite. You may need to add some characters at the end of the name to make it unique.
5. Click Review + Create, then Create.

### **Configure the storage account for static site hosting:**

1. Wait for your storage account to finish being created. When viewing the resource group, you can click Refresh. When your storage account appears in the list, it has been created.
2. Click on your storage account, then click Static Website on the left side of the storage account pane.
3. Click Enabled to enable static website hosting for the storage account.
4. For Index document name, enter index.html.
5. Click Save.

### **Upload the static site files:**

1. Log in to Azure Cloud Shell at https://shell.azure.com.
2. Select Bash, then advanced settings. Select use existing for all three fields, and enter cloudshell for the container.
3. Get the key for the storage account. Note that you can find the resource group name in the Azure portal. The storage account name is the name of the storage account you created earlier.

export AZURE\_STORAGE\_ACCOUNT="&ltstorage account name>"

export RESOURCE\_GROUP="&ltresource group name>"

az storage account keys list

--account-name $AZURE\_STORAGE\_ACCOUNT

--resource-group $RESOURCE\_GROUP

--output table

1. Set an environment variable containing the storage account key.

export AZURE\_STORAGE\_KEY="&ltkey>"

1. Clone the lab resources repository from GitHub and switch to the static site directory.

git clone https://github.com/linuxacademy/content-azurestoragedd-lab-resources.git

cd content-azurestoragedd-lab-resources/satt-static-site/

1. Upload the static site files as blobs within the $web container.

az storage blob upload-batch -d "\$web" -s .

1. In Azure portal, navigate back to the static site storage account and select Static Website again. This page contains a URL under Primary Endpoint. Copy this URL and access it in a new tab.

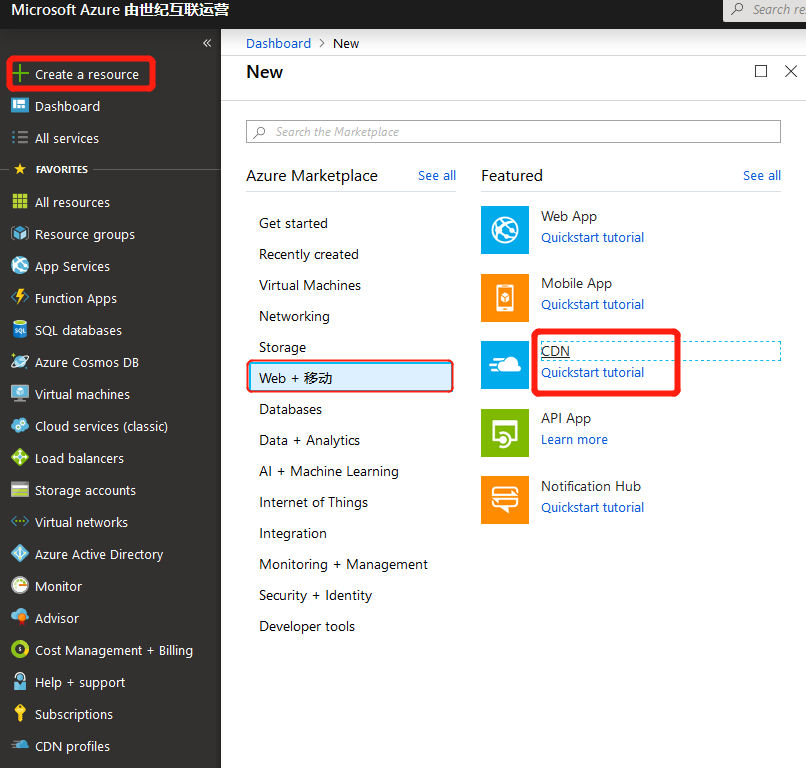
You should see the website for Store All The Things!

**Creating Content delivery network in Azure:**

Step 1: Create a storage account, cloud service, web application

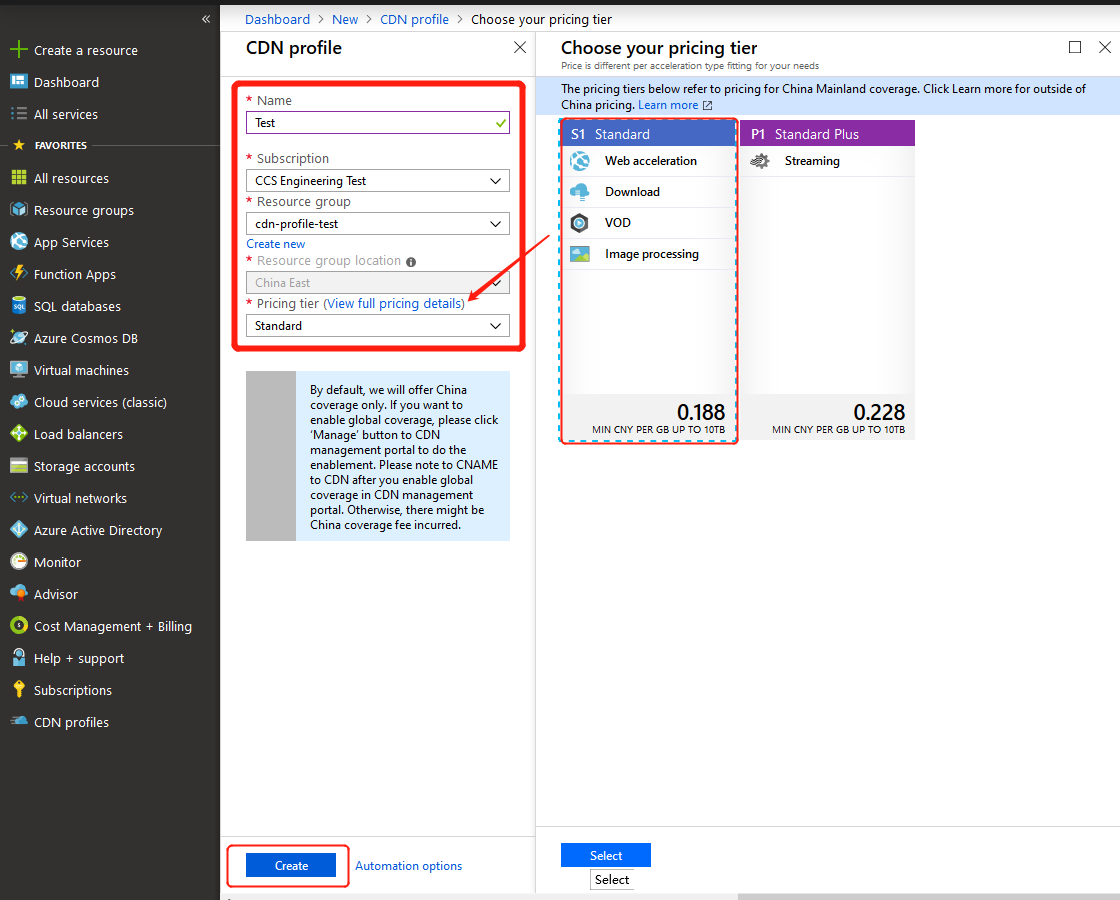
Step 2: Create a CDN Profile resource

* Click Create Resource in the Azure Management Portal, select Web+Mobile, then select CDN Profile



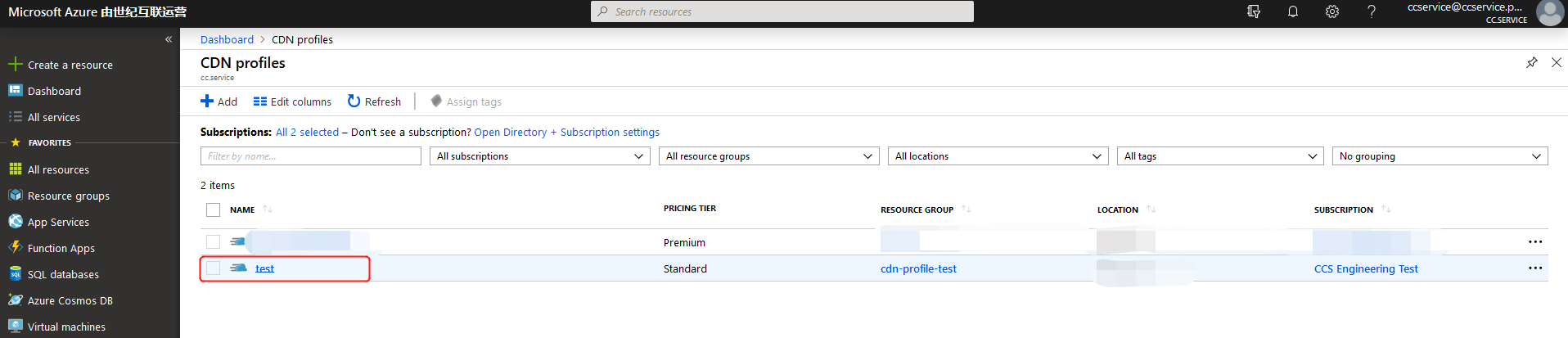
* Fill in the required information and click Create

1. Subscription Select a subscription
2. Resource group Select an existing resource (if no resources have been created, select Create new)
3. Resource group location Defaults to the automatically-selected region
4. Pricing tier Please select “S1 Standard” (“P1 Premium” refers to refers to customers who requested that Azure CDN applies for certificates on their behalf, are still being served using this certificate, and are still paying for the premium version of the CDN).

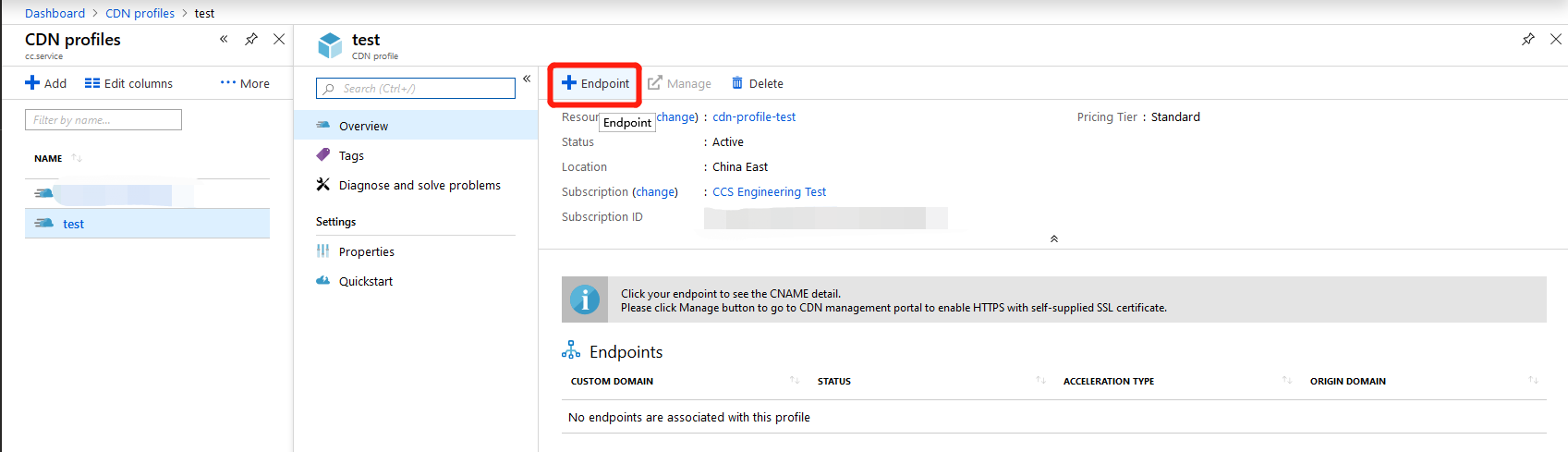


Step 3: Create a new CDN EndPoint node

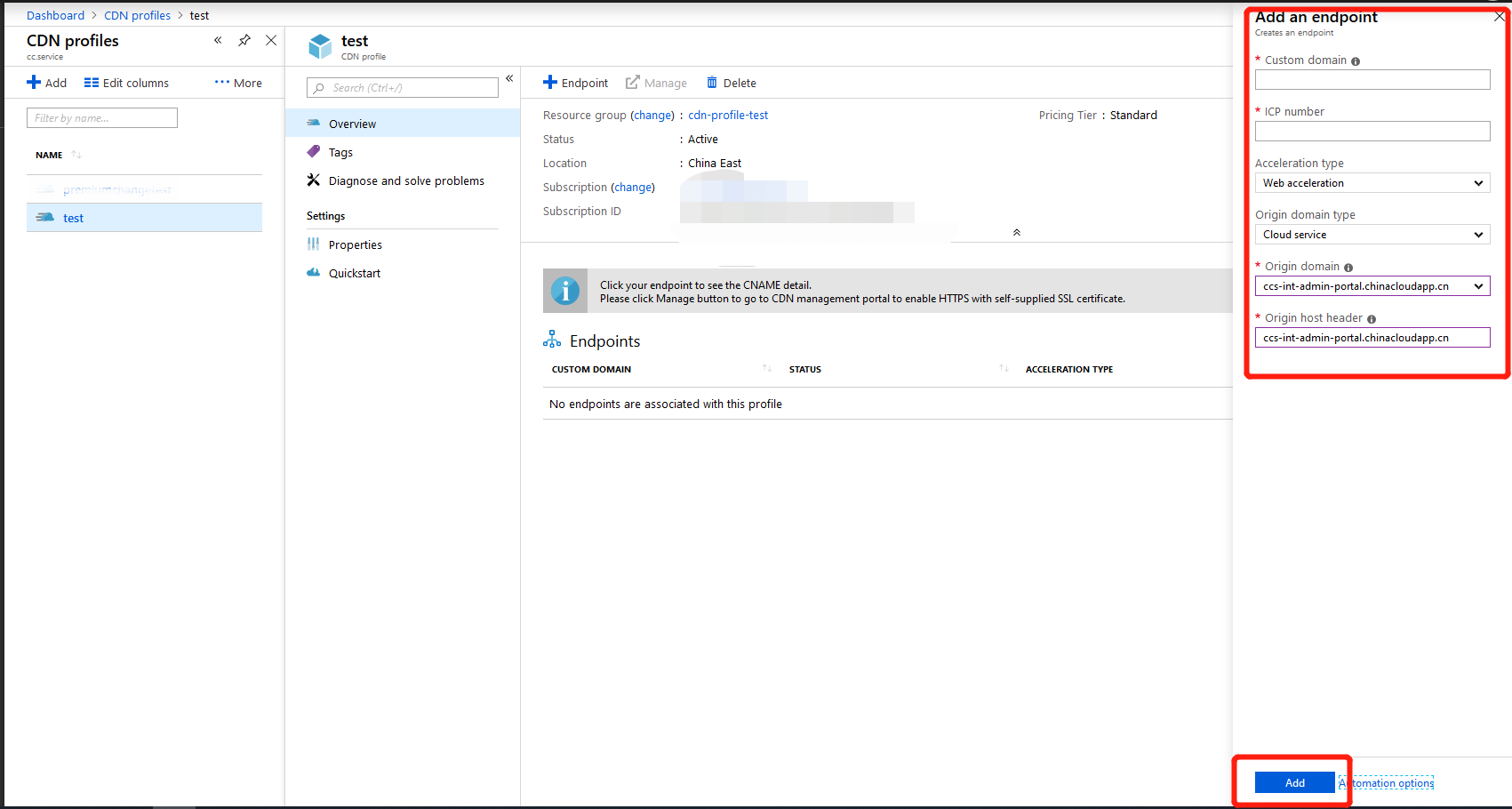
* Find the created Profiles record in the CDN Profiles list, then click on the name



* Click EndPoint



* Enter the required fields ('\*' denotes a required field) and click [Add] to save
  + Customer domain: The domain name requiring acceleration.
  + Acceleration Type: Acceleration type, divided into web/download/vod/live streaming/image processing.
* Web acceleration: Generally used for domains with a large number of images and other files\*
* Download acceleration: Generally used for large files such as apk installation packages and zip archives;
* Vod acceleration: For video on demand;
* Image processing: For acceleration of specific image domain names. Used with domain names that need to be cropped, watermarked, etc.
* Origin domain type: The origin domain type is the Azure service or the customer's origin domain;
* Origin domain: The origin domain can be a domain name or IP address. It can also be multiple IP addresses, but for multiple IP addresses only polling can be used; active-standby is not supported. Active-standby return-to-source must be set offline.
* Origin Host Header: Set the domain name return-to-source. Only domain names can be used. IP addresses usually aren’t used. In general, the origin host header is the same as the accelerated domain or origin domain.



Step 4: Use advanced management features

Once you’ve created a CDN EndPoint, the Advanced Management Portal not only allows you to perform domain name management such as "disable/enable" and "delete" for CDN EndPoint and handle cache (cache refresh) warmup (content prefetch) operations, but also offers features such as Traffic Report, Bandwidth Report, Cache Refresh, Log Download, and Service Check.

* Find the created Profiles record in the CDN Profiles list and click on the name
* Click Manage in Overview

**Migrating from AWS ElasticSearch to ElasticSearch Search on elastic cloud:**

AWS ES clusters are commonly provisioned into a VPC, but they can also be put on a public-facing endpoint. In order to keep this guide universal for both scenarios, we use the Python AWS SDK. You can use any language that has an AWS SDK (e.g., Java, Ruby, Go, etc.) but we only provide examples in Python below.

There are two parts to this guide:

* Part one: Snapshot your AWS ES cluster to S3
* Part two: Restore from S3 to your Elastic Cloud account

Note: If you already have your AWS ES cluster manually snapshotted to S3, you can skip to part two.

Before we begin, it’s important to understand some of the IAM security steps that follow. First, in order to snapshot an AWS ES cluster into S3, your AWS ES cluster needs permission to write to a private S3 bucket. This requires an IAM role and policy that have the necessary permissions. Also, we’ll need to attach an IAM policy to an IAM user (creating one if necessary). The IAM user is used by our script to talk to your AWS ES cluster and by your Elastic-managed deployment to read the snapshot from your S3 bucket.

## **Part one: Snapshot your AWS ES cluster to S3**

### **Step 1: Get your AWS ES info**

We’ll need some basic information about your AWS ES cluster to snapshot it to S3.

* In your AWS Console, go to the Elasticsearch Service
* Select the domain of the cluster you want to snapshot
* Copy the “Domain ARN” value to your notes file (DOMAIN\_ARN)
* Copy the “Endpoint” URL value to your notes file (ES\_ENDPOINT)
* Note which AWS region (e.g., us-east-1) your AWS ES cluster is in (ES\_REGION)

This information is used below in the IAM policy creation and when it’s time to issue commands to the cluster.

### **Step 2 - Create an AWS S3 bucket**

We need to create an S3 bucket to store your snapshot.

Important: Your S3 bucket must be in the same region as your AWS ES cluster. You will be able to restore from there to an Elastic-managed deployment in any region or cloud provider (**AWS**, **GCP**, or **Azure**).

* In your AWS Console, go to the S3 service
* Create a private S3 bucket
* Note: If you leave the defaults, your bucket will be private and secure
* Copy the name of the bucket to your notes file (S3\_BUCKET\_NAME)
* Copy the region of the bucket to your notes file (S3\_REGION\_NAME)

This information is used when we register a snapshot repository with Elasticsearch.

### **Step 3 - Create an IAM role**

Next we’ll create a role to delegate permission to Amazon Elasticsearch Service to take a snapshot into S3.

* In your AWS Console, go to the IAM service
* Select “Roles”
* Select “Create role”
* Select “EC2” as the service that will use this new role (we will change it later)
* Select “Next: Permissions”
* Leave the policies on the role empty for now
* Select “Next: Tags”
* Select “Next: Review”
* Name the role: TheSnapshotRole
* Select “Create role”
* From the list of roles, select the one we just created: TheSnapshotRole
* Select “Trust relationships”
* Select “Edit trust relationship”
* Copy and paste the following into the trust relationship (replacing what’s there)

{

"Version": "2012-10-17",

"Statement": [{

"Effect": "Allow",

"Principal": {

"Service": "es.amazonaws.com"

},

"Action": "sts:AssumeRole"

}]

}

* Select “Update Trust Policy”
* Select “Permissions”
* Select “Add inline policy”
* Select the JSON tab
* Copy and paste the following JSON (replacing what’s there)
* Replace S3\_BUCKET\_NAME with the correct value (in two places)

{

"Version": "2012-10-17",

"Statement": [{

"Action": [

"s3:ListBucket"

],

"Effect": "Allow",

"Resource": [

"arn:aws:s3:::S3\_BUCKET\_NAME"

]

},

{

"Action": [

"s3:GetObject",

"s3:PutObject",

"s3:DeleteObject"

],

"Effect": "Allow",

"Resource": [

"arn:aws:s3:::S3\_BUCKET\_NAME/\*"

]

}

]

}

* Select “Review policy”
* Name the policy: TheSnapshotS3Policy
* Select “Create policy”
* Copy the “Role ARN” value to your notes file (ROLE\_ARN)

We just created an IAM role with an inline policy that can read & write to your S3 bucket.

### **Step 4 - Create an IAM policy**

We need to create a new IAM policy that has permission to assume the role above in order to register the snapshot repository.

* In your AWS Console, go to the IAM service
* Select “Policies”
* Select “Create policy”
* Select the JSON tab
* Copy and paste the following JSON (replacing what’s there)
* Replace ROLE\_ARN with the correct value
* Replace DOMAIN\_ARN with the correct value
* Replace S3\_BUCKET\_NAME with the correct value (in 2 places)

{

"Version": "2012-10-17",

"Statement": [

{

"Effect": "Allow",

"Action": "iam:PassRole",

"Resource": "ROLE\_ARN"

},

{

"Effect": "Allow",

"Action": "es:ESHttpPut",

"Resource": "DOMAIN\_ARN/\*"

}

]

}

* Select “Review policy”
* Name the policy: TheSnapshotPolicy
* Select “Create policy”

We just created an IAM policy that allows the IAM role to talk to your AWS ES domain.

### **Step 5 - Create an IAM user**

If you don’t already have an IAM user, we’ll need to create one and give it access to your private S3 bucket. If you do have an IAM user, you can simply attach the following IAM policy to it.

* In your AWS Console, go to the IAM service
* Select “Users”
* Select “Add user”
* Name the user: TheSnapshotUser
* Check the box “Programmatic access”
* Select “Next: Permissions”
* Select the box “Attach existing policies directly”
* Filter the policies by typing in “TheSnapshot”
* Select the checkbox next to the policy “TheSnapshotPolicy”
* Select “Next: Tags”
* Select “Next: Review”
* Select “Create user”
* Copy the “Access key ID” value to your notes file (ACCESS\_KEY)
* Select “Show” under “Secret access key”
* Copy the “Secret access key” value to your notes file (SECRET\_KEY)
* Select “Close”
* From the list of users, select the one we just created: TheSnapshotUser
* Select “Add inline policy”
* Select the JSON tab
* Copy and paste the following JSON (replacing what’s there)
* Replace S3\_BUCKET\_NAME with the correct value (in 2 places)

{

"Version": "2012-10-17",

"Statement": [{

"Action": [

"s3:ListBucket"

],

"Effect": "Allow",

"Resource": [

"arn:aws:s3:::S3\_BUCKET\_NAME"

]

},

{

"Action": [

"s3:GetObject",

"s3:PutObject",

"s3:DeleteObject"

],

"Effect": "Allow",

"Resource": [

"arn:aws:s3:::S3\_BUCKET\_NAME/\*"

]

}

]

}

* Select “Review policy”
* Name the policy: TheSnapshotUserS3Policy
* Select “Create policy”

We just created an IAM user that can take a manual snapshot and read from that snapshot.

### **Step 6 - Configure the Python AWS SDK**

Installing pip3

To install pip3 on Red Hat and derivatives, use yum:

$ sudo yum -y install python3-pip

Alternatively, some Fedora distributions label the pip3 package differently:

$ sudo yum -y install python36-pip

You can search for it if neither package name works above:

$ yum search pip

On Debian derivatives such as Ubuntu, use apt-get:

$ sudo apt-get -y install python3-pip

**Installing the Python AWS SDK**

Once pip3 is installed, you can install the Python AWS SDK called boto3:

$ pip3 install --user boto3 requests\_aws4auth

Collecting boto3

…

Successfully installed boto3-1.9.106 requests-aws4auth-0.9 ...

Note: No root access is needed if you specify the --user flag.

We need to create an ~/.aws directory to hold our AWS credentials. Run the following command to create the directory:

$ mkdir ~/.aws

Create a file called credentials with your favorite editor. We’ll use nano for simplicity:

$ nano ~/.aws/credentials

Copy and paste the following contents into the file, replacing the 2 uppercase variables.

aws\_access\_key\_id = ACCESS\_KEY

aws\_secret\_access\_key = SECRET\_KEY

### **Step 7 - Manually snapshot AWS ES**

Let’s run a quick test using a Python script to list the indices in our AWS ES cluster. This will ensure our AWS credentials are working and prove we can talk to the cluster.

Create a file called indices.py with your favorite editor. We’ll use nano for simplicity:

$ nano indices.py

Copy and paste the following contents, replacing the two uppercase variables with your values:

import boto3, requests

from requests\_aws4auth import AWS4Auth

host = 'ES\_ENDPOINT'

region = 'ES\_REGION'

creds = boto3.Session().get\_credentials()

auth = AWS4Auth(creds.access\_key, creds.secret\_key, region, 'es', session\_token=creds.token)

print("Listing Indices from AWS ES ...")

req = requests.get(host + '/\_cat/indices?v', auth=auth)

print("HTTP Response Code: " + str(req.status\_code) + '\n' + req.text)

Use ctrl+x to exit nano, and follow the prompts to save the file.

Run the Python script.

$ python3 indices.py

Your output should look similar to the following:

Listing Indices from AWS ES …

HTTP Response Code: 200

health status index uuid pri rep docs.count docs.deleted store.size pri.store.size

green open testindex yME2BphgR3Gt1ln6n03nHQ 5 1 1 0 4.4kb 4.4kb

Now create a file called register.py with your favorite editor.

$ nano register.py

Copy and paste the following contents, replacing the seven uppercase variables with your values:

import boto3, requests

from requests\_aws4auth import AWS4Auth

import boto3, requests

from requests\_aws4auth import AWS4Auth

host = 'ES\_ENDPOINT'

region = 'ES\_REGION'

repo\_name = 'SNAPSHOT\_REPO'

snapshot\_name = 'SNAPSHOT\_NAME'

s3\_region\_name = 'S3\_REGION\_NAME'

s3\_bucket\_name = 'S3\_BUCKET\_NAME'

role\_arn = 'ROLE\_ARN'

creds = boto3.Session().get\_credentials()

auth = AWS4Auth(creds.access\_key, creds.secret\_key, region, 'es', session\_token=creds.token)

headers = {"Content-Type": "application/json"}

payload = {

"type": "s3",

"settings": {

"region": s3\_region\_name,

"bucket": s3\_bucket\_name,

"role\_arn": role\_arn

}

}

print("Registering Snapshot with AWS ES ...")

url = host + '/\_snapshot/' + repo\_name

req = requests.put(url, auth=auth, json=payload, headers=headers)

print("HTTP Response Code: " + str(req.status\_code) + '\n' + req.text)

Use ctrl+x to exit nano, and follow the prompts to save the file.

Run the Python script.

$ python3 register.py

Your output should look similar to the following:

Registering Snapshot with AWS ES …

HTTP Response Code: 200

{"acknowledged":true}

Next, create a file called snapshot.py with your favorite editor.

$ nano snapshot.py

Copy and paste the following contents, replacing the four upper variables with your values:

import boto3, requests

from requests\_aws4auth import AWS4Auth

host = 'ES\_ENDPOINT'

region = 'ES\_REGION'

repo\_name = 'SNAPSHOT\_REPO'

snapshot\_name = 'SNAPSHOT\_NAME'

creds = boto3.Session().get\_credentials()

auth = AWS4Auth(creds.access\_key, creds.secret\_key, region, 'es', session\_token=creds.token)

print("Starting Snapshot with AWS ES ...")

url = host + '/\_snapshot/' + repo\_name + '/' + snapshot\_name

req = requests.put(url, auth=auth)

print("HTTP Response Code: " + str(req.status\_code) + '\n' + req.text)

Use ctrl+x to exit nano, and follow the prompts to save the file.

Run the Python script.

$ python3 snapshot.py

Your output should look similar to the following:

Starting Snapshot with AWS ES …

HTTP Response Code: 200

{"accepted":true}

Finally, let’s check the status of our snapshot. Create a file called status.py.

$ nano status.py

Copy and paste the following contents, replacing the four upper variables with your values:

import boto3, requests

from requests\_aws4auth import AWS4Auth

host = 'ES\_ENDPOINT'

region = 'ES\_REGION'

repo\_name = 'SNAPSHOT\_REPO'

snapshot\_name = 'SNAPSHOT\_NAME'

creds = boto3.Session().get\_credentials()

auth = AWS4Auth(creds.access\_key, creds.secret\_key, region, 'es', session\_token=creds.token)

print("Getting Status of Snapshot with AWS ES ...")

url = host + '/\_snapshot/' + repo\_name + '/' + snapshot\_name + '?pretty'

req = requests.get(url, auth=auth)

print("HTTP Response Code: " + str(req.status\_code) + '\n' + req.text)

Run the Python script.

$ python3 status.py

Your output should look similar to the following:

Getting Status of Snapshot with AWS ES ...

HTTP Response Code: 200

{

"snapshots" : [ {

"snapshot" : "my-snapshot",

"uuid" : "ClYKt5g8QFO6r3kTCEzjqw",

"version\_id" : 6040299,

"version" : "6.4.2",

"indices" : [ "testindex" ],

"include\_global\_state" : true,

"state" : "SUCCESS",

"start\_time" : "2019-03-03T14:46:04.094Z",

"start\_time\_in\_millis" : 1551624364094,

"end\_time" : "2019-03-03T14:46:04.847Z",

"end\_time\_in\_millis" : 1551624364847,

"duration\_in\_millis" : 753,

"failures" : [ ],

"shards" : {

"total" : 5,

"failed" : 0,

"successful" : 5

}

} ]

}

## **Part two – Restore from S3**

### **Step 1 - Size your deployment**

The deployment you created in Elasticsearch Service on Elastic Cloud should have the same amount of resources as your AWS ES cluster. Use the sliders and increment the number of data nodes to reflect the size of the cluster you have in AWS ES. Save your changes before proceeding.

### **Step 2 - Add a custom repository**

In your Elastic-managed deployment (not your AWS ES cluster), open Kibana and go to “Dev Tools”.

Copy and paste the following API call into Dev Tools, replacing the five variables:

PUT /\_snapshot/SNAPSHOT\_REPO

{

"type": "s3",

"settings": {

"bucket": "S3\_BUCKET\_NAME",

"region": "S3\_REGION\_NAME",

"access\_key": "ACCESS\_KEY",

"secret\_key": "SECRET\_KEY",

"compress": true

}

}

You should get the following response:

{

"acknowledged": "true"

}

### **Step 3 - Restore from S3**

Finally, it’s time to restore from the snapshot repository we just registered.

Copy and paste the following API call into Dev Tools, replacing the two variables:

POST /\_snapshot/SNAPSHOT\_REPO/SNAPSHOT\_NAME/\_restore

You should get the following response:

{

"accepted": "true"

}

You can check the progress of your restore with the following:

GET /\_snapshot/SNAPSHOT\_REPO/SNAPSHOT\_NAME/\_status

If you see "state":"SUCCESS" your restore completed successfully:

{

"snapshots": [

{

"snapshot": "my-snapshot",

"repository": "my-snapshot-repo",

"state": "SUCCESS",

...

}

]

}

**Creating functions in Azure:**

## **Create a function app:**

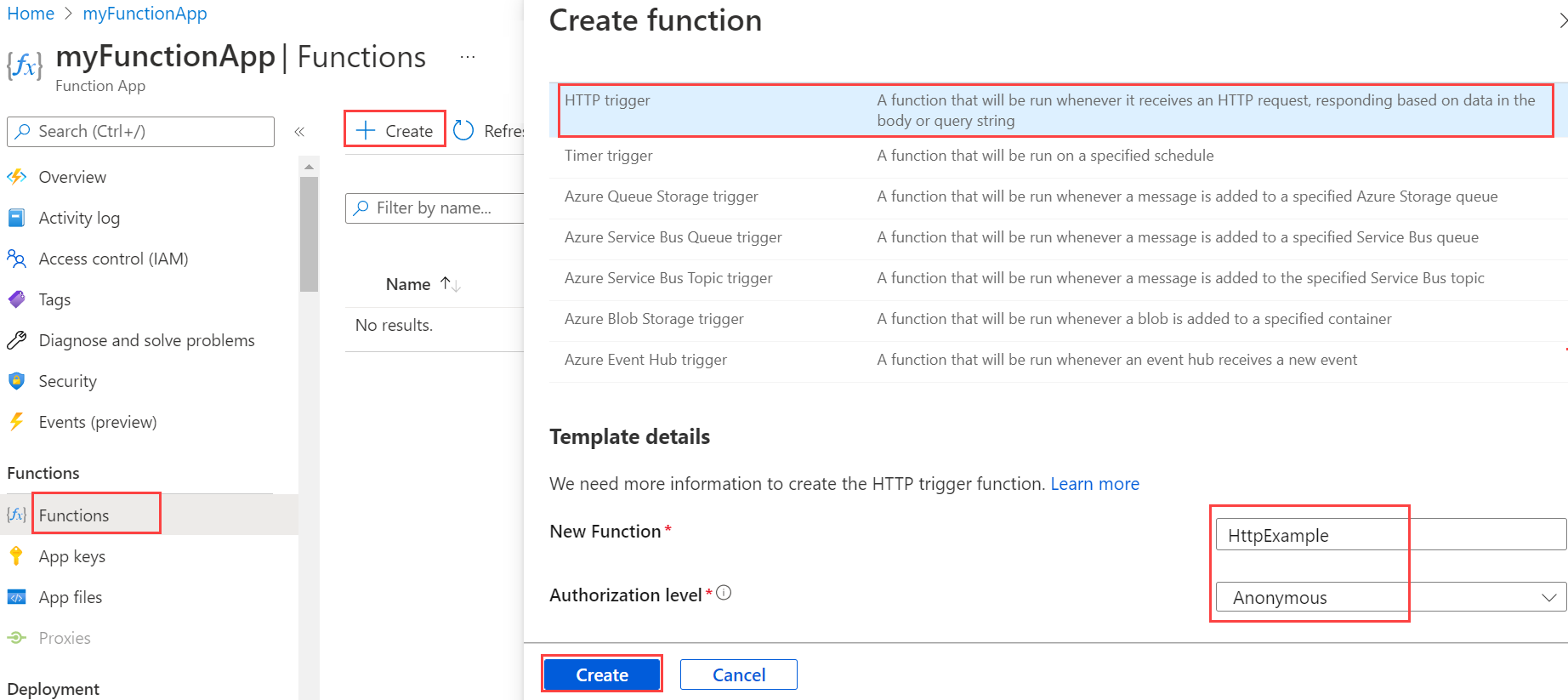
You must have a function app to host the execution of your functions. A function app lets you group functions as a logical unit for easier management, deployment, scaling, and sharing of resources.

1. From the Azure portal menu or the Home page, select Create a resource.
2. In the New page, select Compute > Function App.
3. Accept the default options of creating a new storage account on the Storage tab and a new Application Insight instance on the Monitoring tab. You can also choose to use an existing storage account or Application Insights instance.
4. Select Review + create to review the app configuration you chose, and then select Create to provision and deploy the function app.

## 

## **Create an HTTP trigger function:**

1. From the left menu of the Function App window, select Functions, and then select Create from the top menu.
2. From the Create Function window, leave the Development environment property as Develop in portal, and then select the HTTP trigger template.

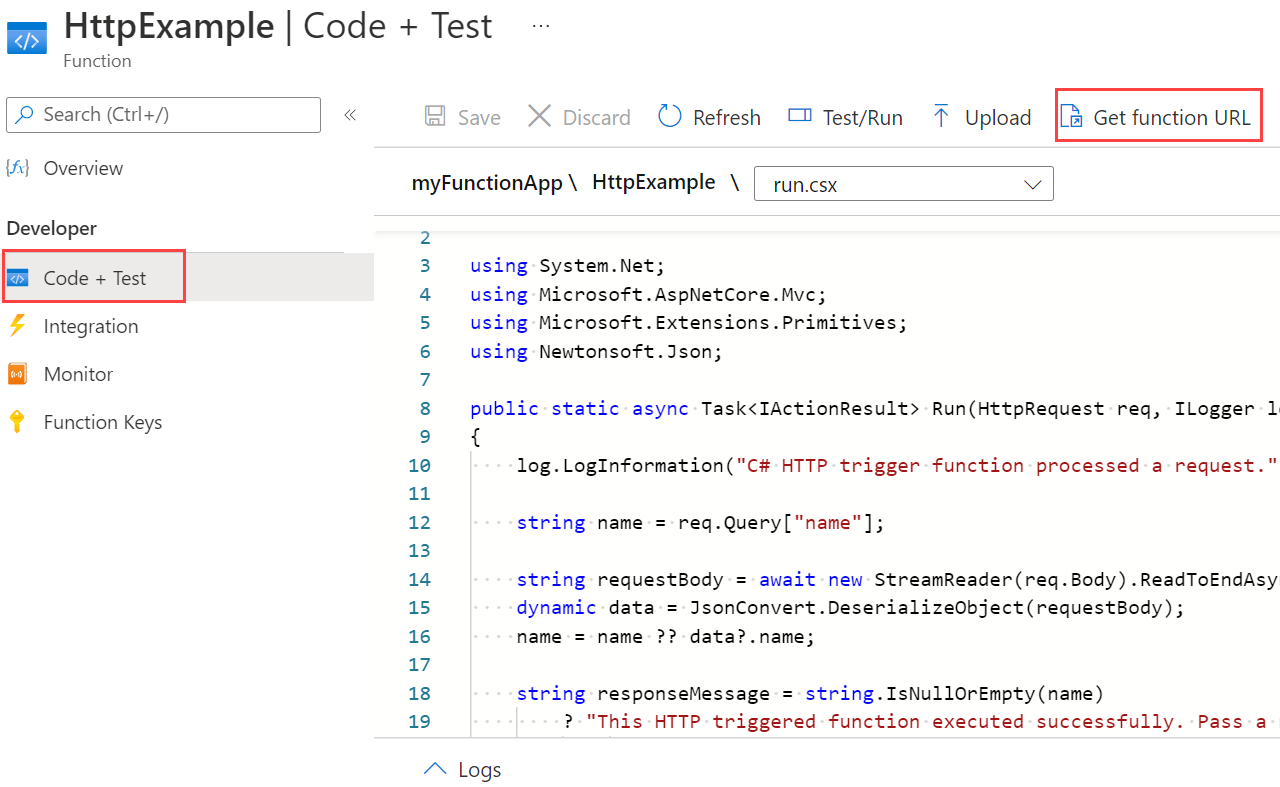


1. Under Template details use HttpExample for New Function, select Anonymous from the Authorization level drop-down list, and then select Create.

Azure creates the HTTP trigger function. Now, you can run the new function by sending an HTTP request.

## **Test the function**

1. In your new HTTP trigger function, select Code + Test from the left menu, and then select Get function URL from the top menu.



1. In the Get function URL dialog, select default from the drop-down list, and then select the Copy to clipboard icon.



1. Paste the function URL into your browser's address bar. Add the query string value ?name=<your\_name> to the end of this URL and press Enter to run the request. The browser must display a response message that echoes back your query string value.

If the request URL included an access key (?code=...), it means you selected Function instead of Anonymous access level when creating the function. In this case, you must instead append &name=<your\_name>.

1. When your function runs, trace information is written to the logs. To see the trace output, return to the Code + Test page in the portal and expand the Logs arrow at the bottom of the page. Call your function again to see the trace output written to the logs.

## **Creating functions through IDE(vscode):**

Azure Functions is the function-as-a-service (FaaS) offering from Azure—the equivalent of AWS Lambda from Amazon.

### **1. Setting Up Your Local Environment**

Even though you can start right away by creating your first function in the portal, I’m going to guide you on how to start as a traditional developer by using an IDE.

I chose VS Code as the IDE and Javascript as the language for the app to demonstrate the versatility of Azure Functions.

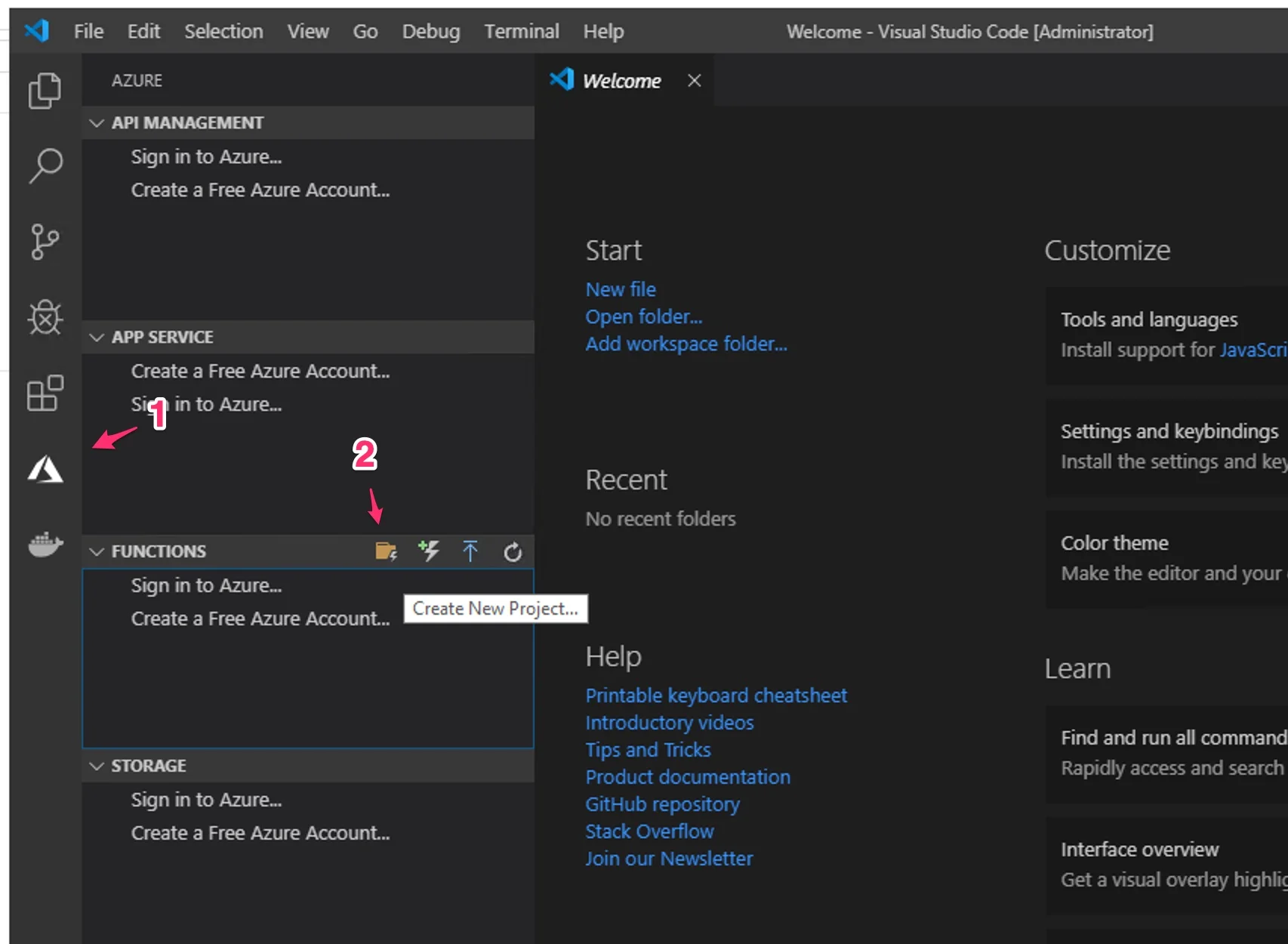
To complete this tutorial, you need to install the following:

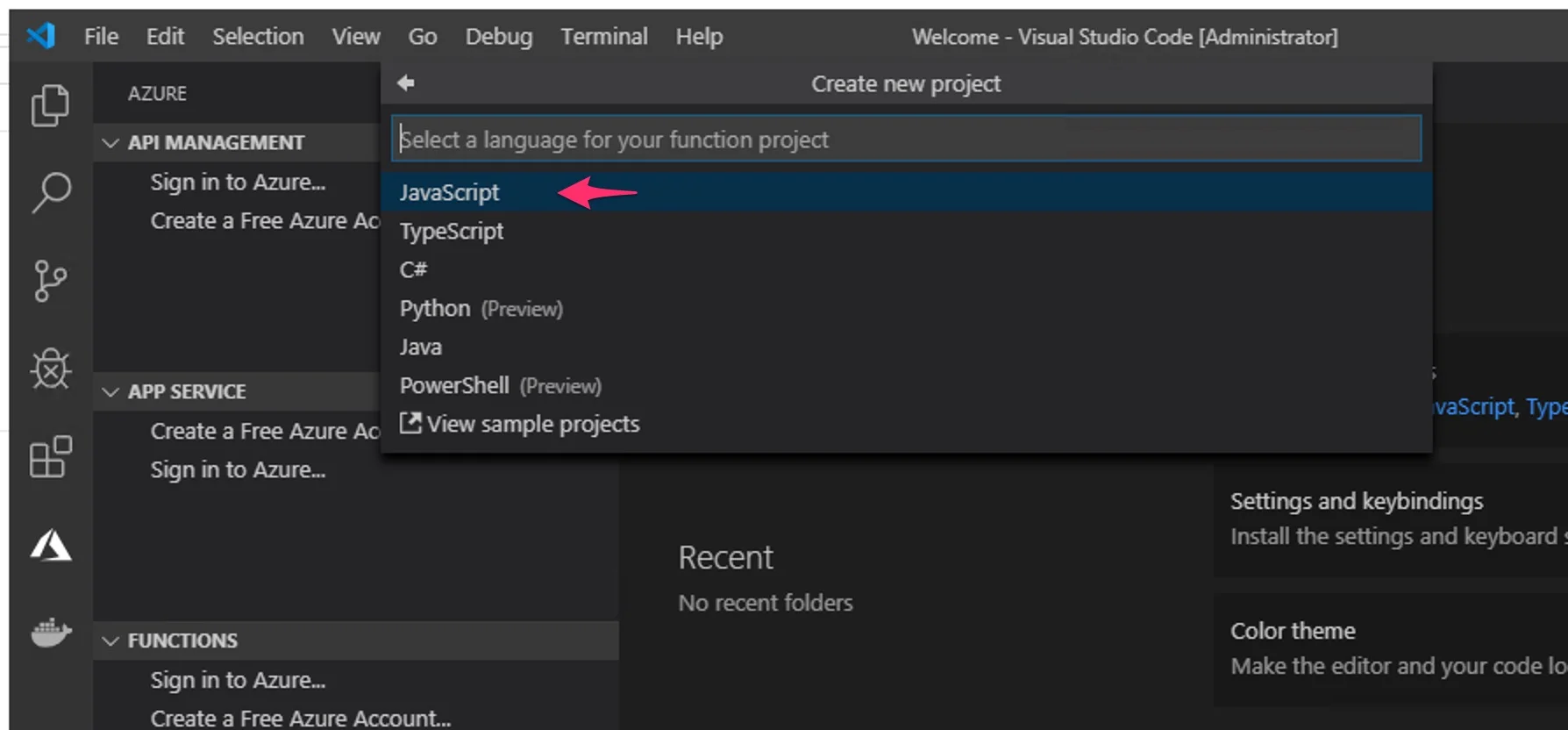
* Node 8.0 or higher.
* VS Code. It’s supported on Windows, Linux, or Mac.
* Azure Tools for VS Code, and sign in to your Azure account.
* Install the Azure Functions Core Tools.

### **2. Create the Azure Functions Project**

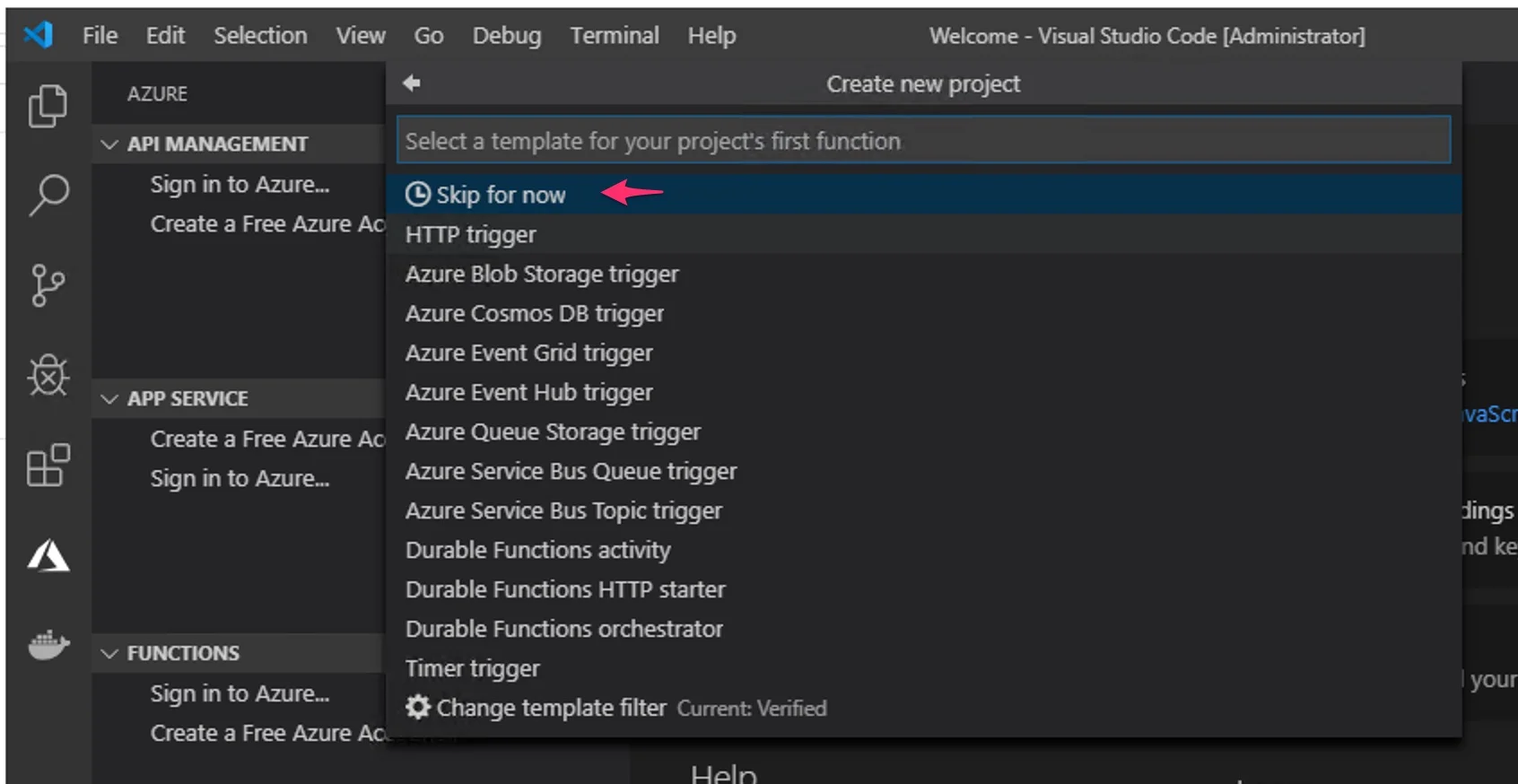
It’s time to create the Azure Functions project in VS Code. Click on the “Azure” logo (1) in the sidebar.

Next, click on the folder icon (2) and choose a location for the project where you don’t have an existing workspace. You can create a new folder if you want.



Now you should see a screen to choose the application code language. Choose Javascript.

Click on “Skip for now” to only create the project.



In the next screen, choose the “Add to workspace” option to finish creating the project.

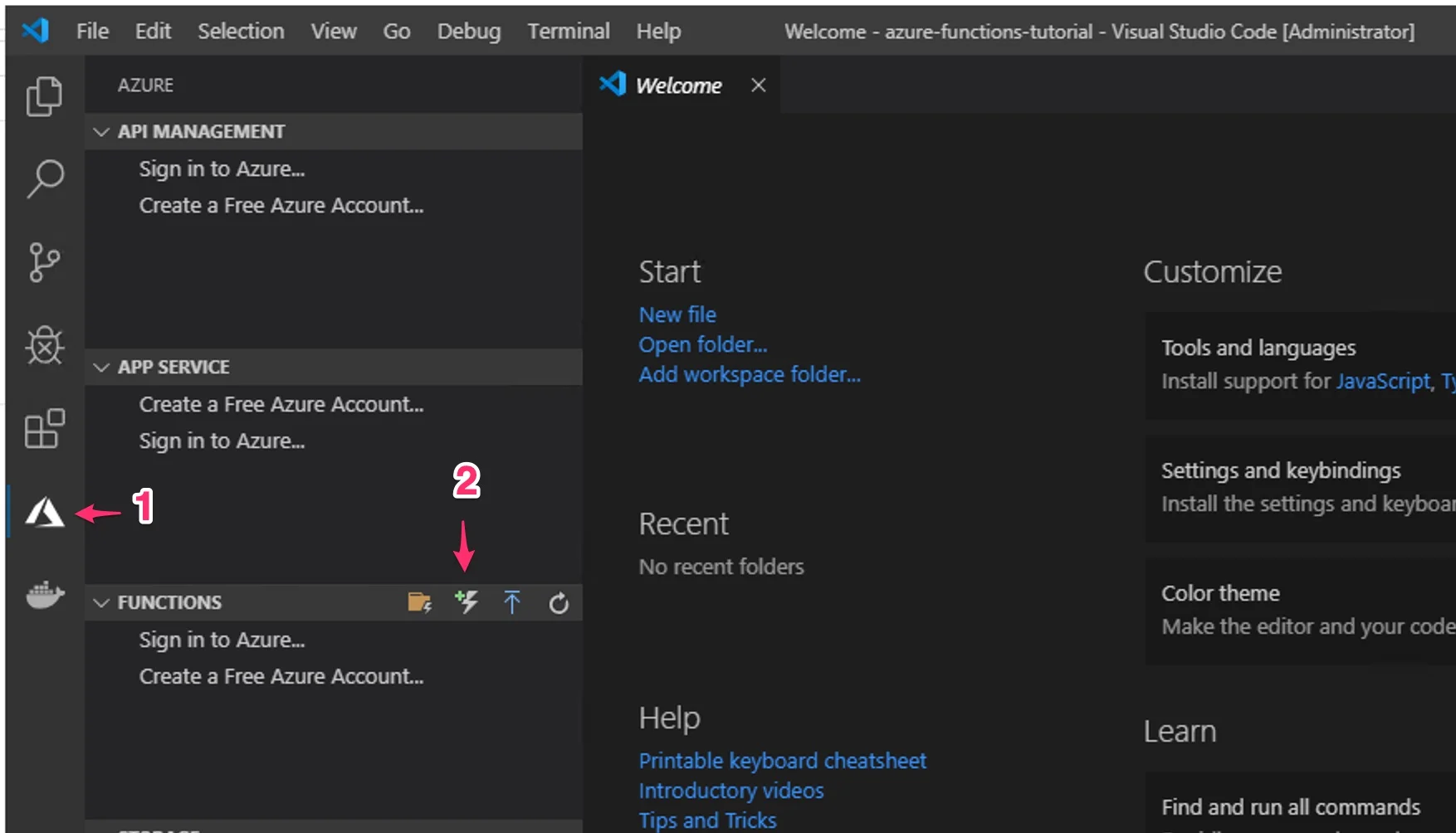
### **3. Create the Azure Function With an HTTP Trigger**

When you upload a function, it won’t do anything by itself—you need to configure a trigger. A trigger is how the function interacts with other services or events.

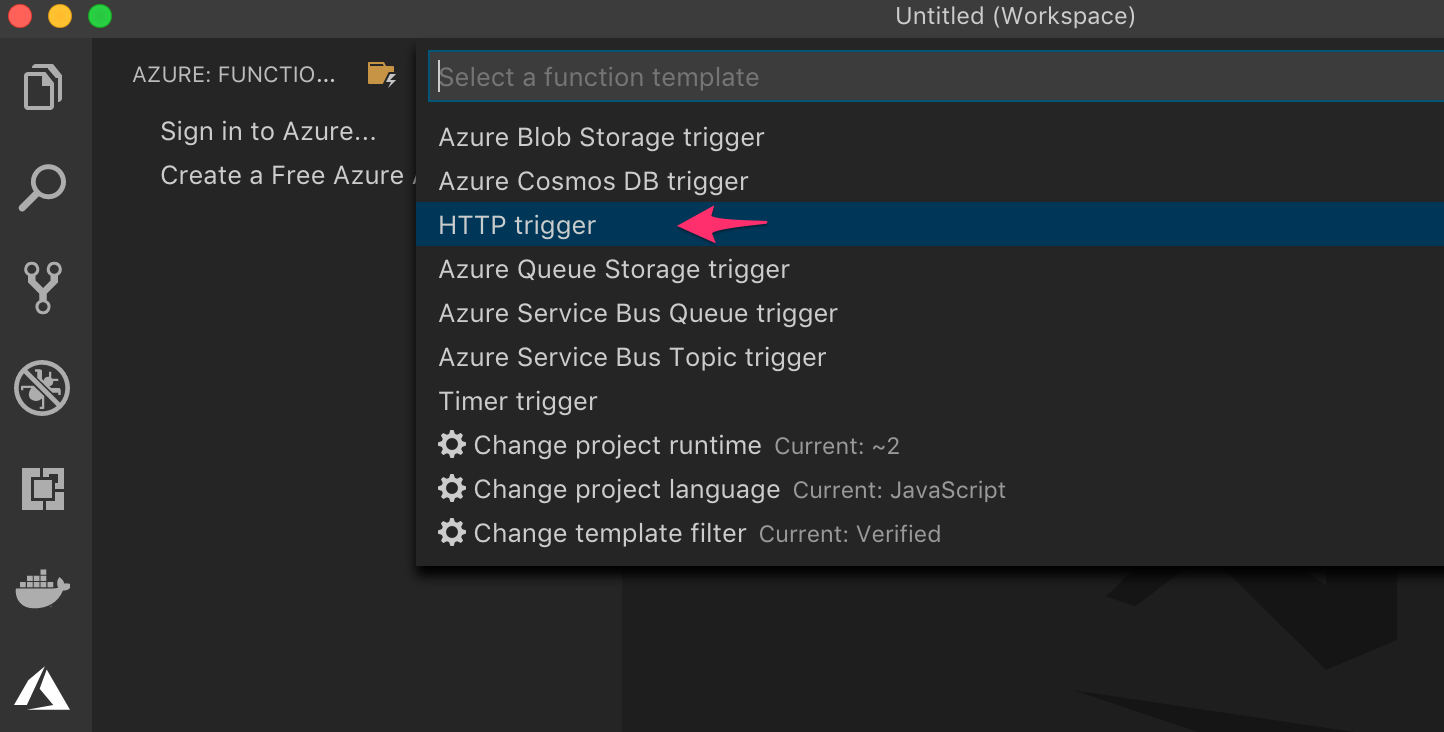
For example, every time you store an item in an Azure Cosmos DB or for every new message in an Azure Service Bus Queue. Today we’re going to keep it simple and create an HTTP trigger.

Azure will generate an HTTP endpoint that you can access to interact with the application. Unlike AWS, where you need to create an API Gateway, Azure Functions creates that for you when you choose the HTTP trigger.

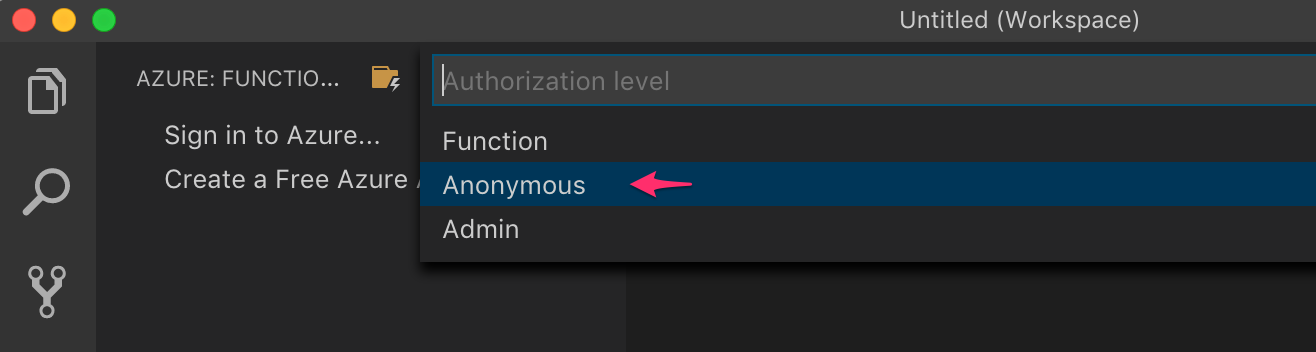
In VS Code, click on the Azure logo in the sidebar (1) then click on the thunder with a plus (2) to create the function.



Choose the folder you just created for the project and click on the “HTTP trigger” option. Type a name for the function and press “Enter” on your keyboard.

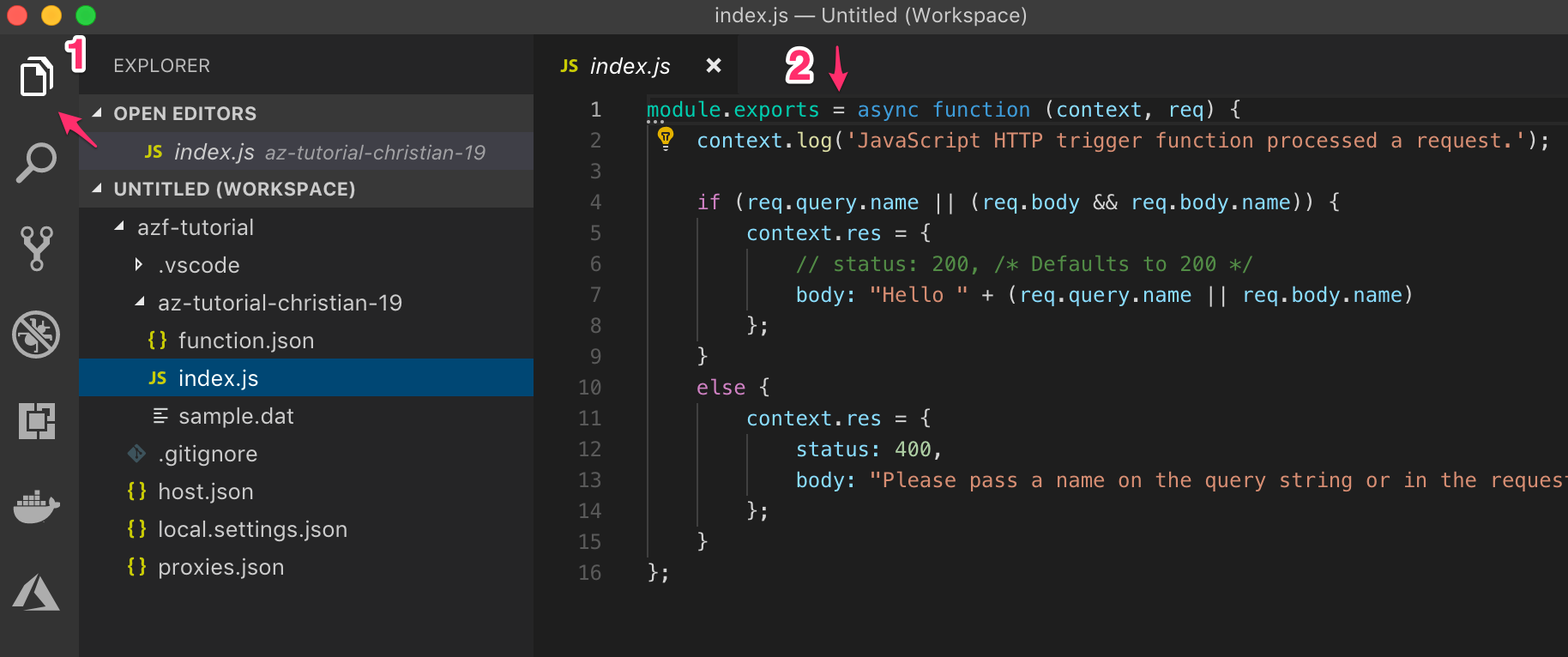


In the following screen, choose the “Anonymous” option to give access to the HTTP endpoint without any authentication method. This step is how you’ll set up the security access for the function.



You should see the code of the first function; you can click on the “Explorer” icon (1) in the sidebar to see the project structure. In the right panel, you’ll see the code (2). Leave it as it is, for now.

You can change it later after you’ve published a working version of this example application in Azure.

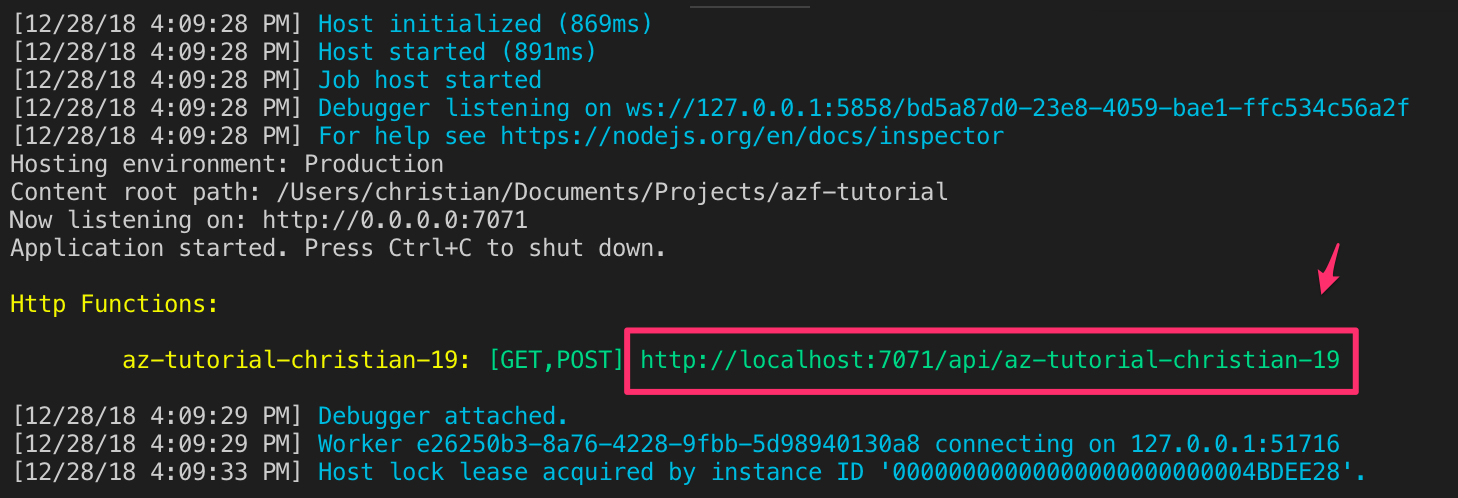


It’s time to test it locally before publishing to Azure.

### **4. Locally Run and Debug the Function**

You can set a breakpoint in any part of the code to see what it’s like to debug the code in VS Code. Hit “F5” and VS Code will initiate the Azure Functions runtime.

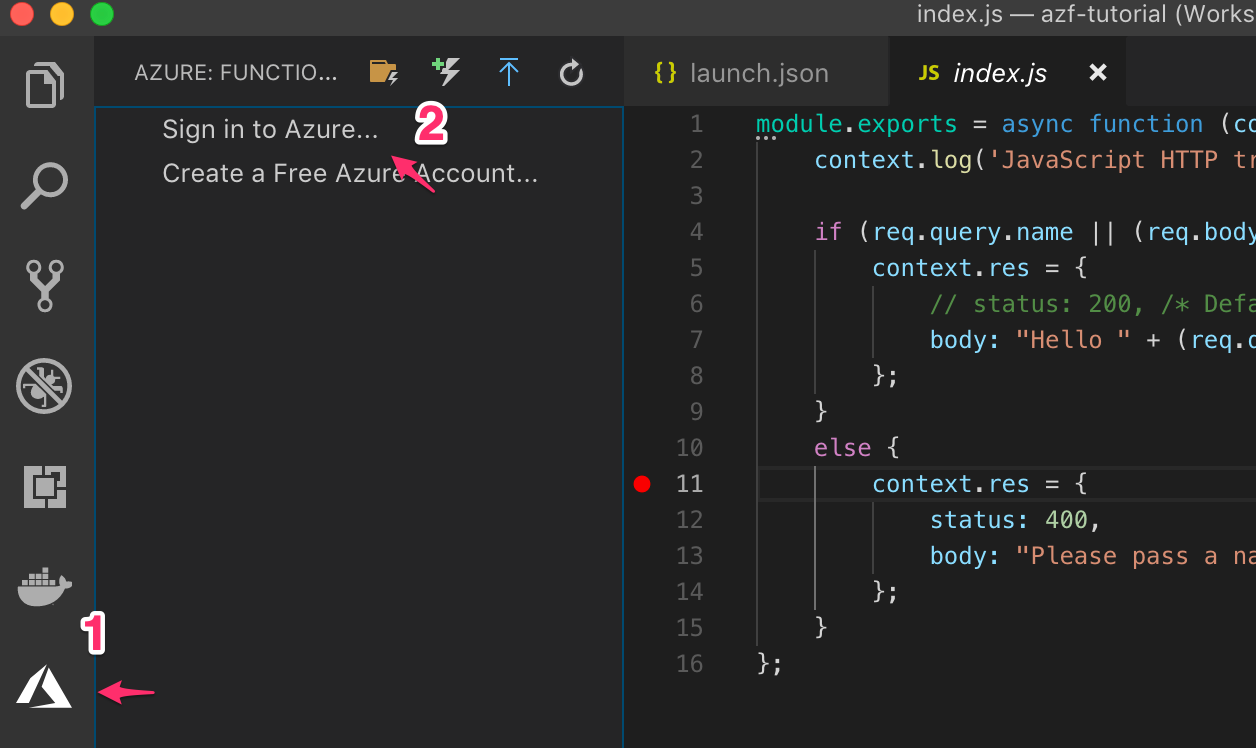
In the console output, you’ll see the local URL to test the application.



### **5. Publish the Function to Azure**

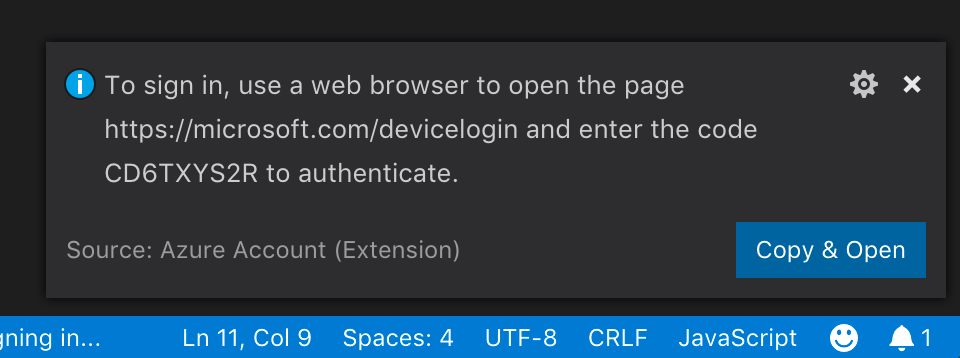
Let’s connect VS Code to Azure now. Even though I wouldn’t recommend publishing directly from VS Code, it’s a good way to start interacting with Azure Functions.

Click on the “Azure” logo in the left menu (1), and then click on the “Sign in to Azure…”

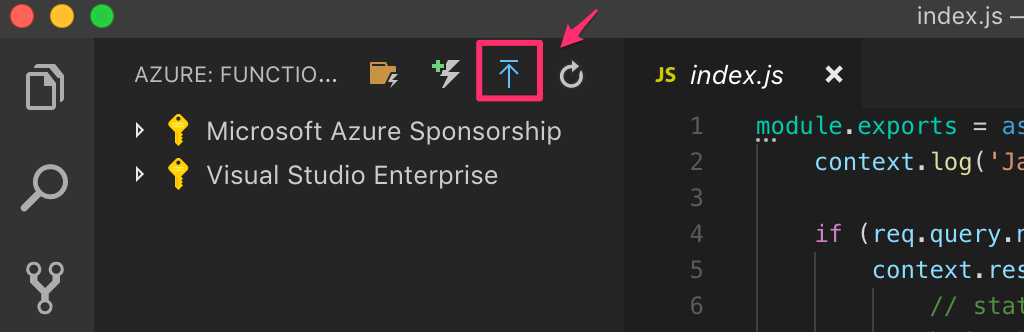


You’ll see a prompt in the bottom-right of VS Code with a sign in URL and a code to authenticate. You can click the blue “Copy & Open” button and a browser window will open.

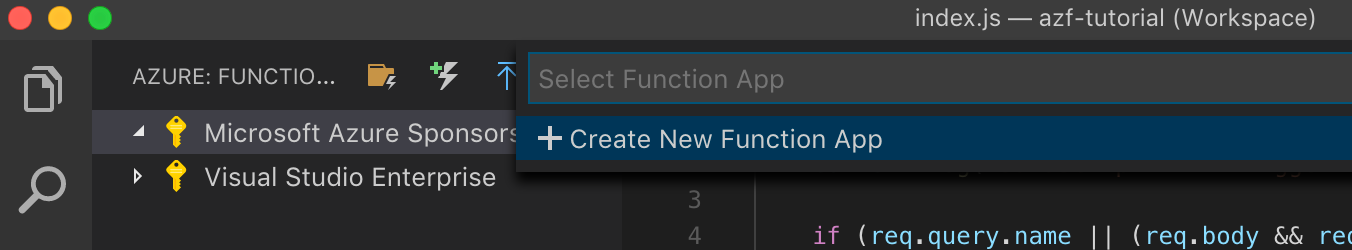
Paste the code and sign in with your browser. VS Code will automatically detect that you’ve signed in successfully and will connect to Azure.



You’ll see that any Azure subscriptions you have will appear in the left panel. Click on the “Deploy to Function App” icon and then select the subscription where you’ll deploy the function.

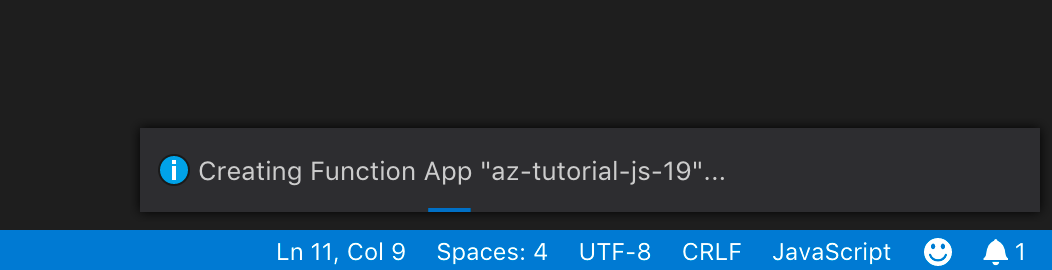


Click on the “+ Create New Function App” option and enter the function app name you want. This name has to be unique across all Azure, so be creative and unique.

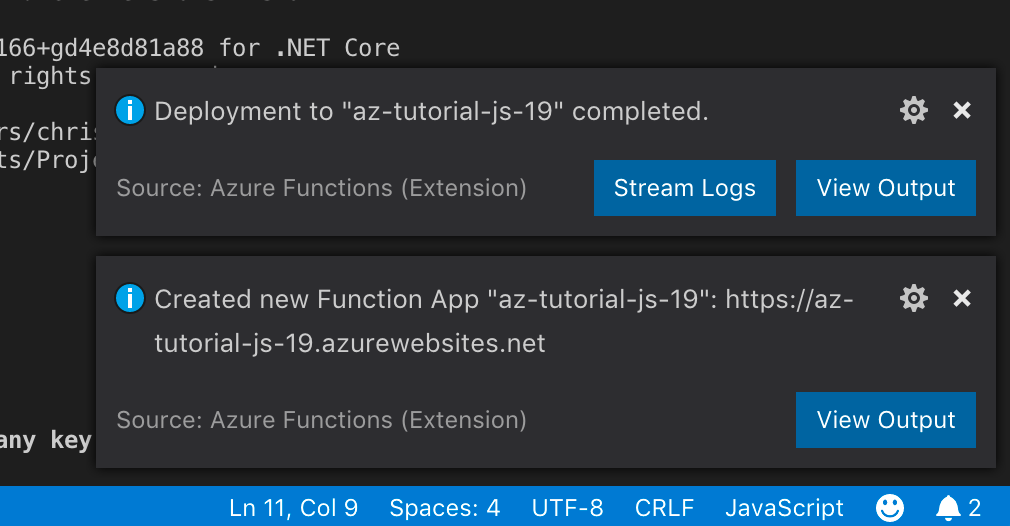


In the following screen, click on the “+ Create new resource group” and enter a unique name across your account. Then click on the “+ Create new storage account” and enter a unique name across your account.

Finally, choose the region where you want to deploy the function and give it some time while the function app is created.



When it’s done, you’ll see in the output the URL of the HTTP endpoint published on Azure.



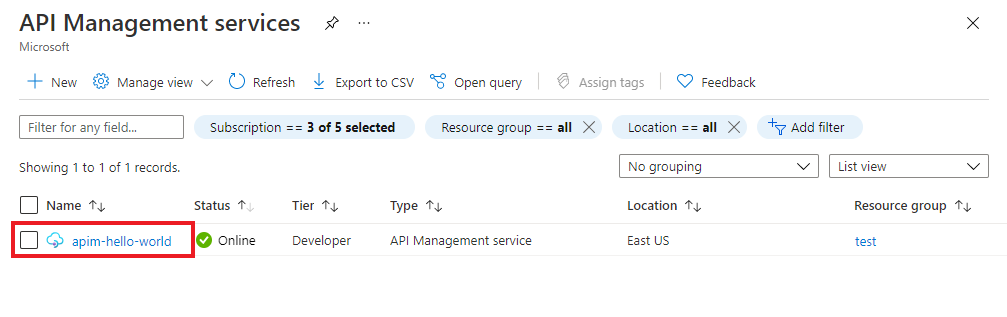
Open the URL in a browser window and test the application again.

**Create API Management service in Azure:**

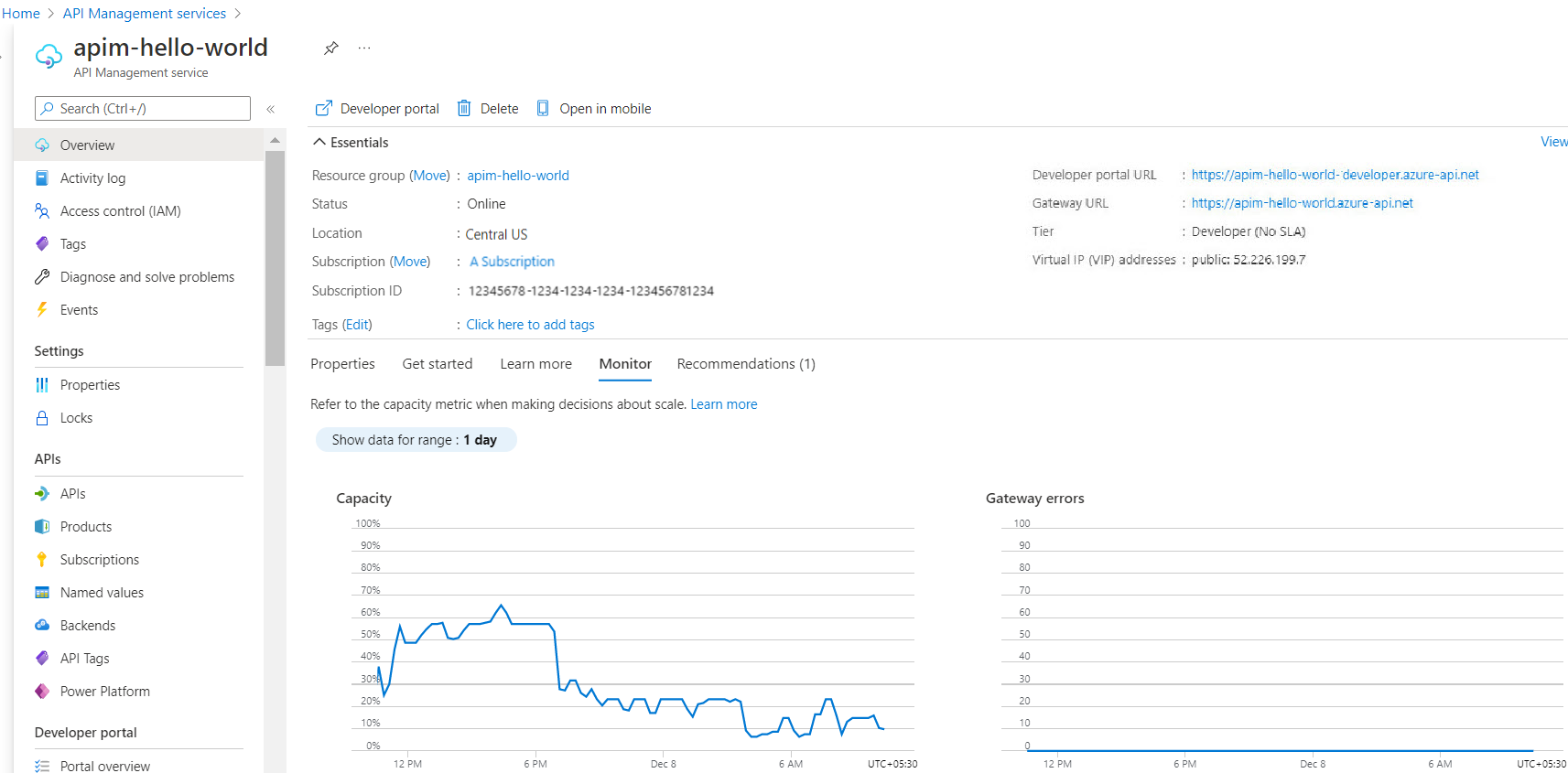
1. Create a new service
2. On the Create a resource page, select Integration > API Management.
3. In the Create API Management page, enter settings.
4. Select Review + create.

**Go to your API Management instance**

1. In the Azure portal, search for and select API Management services.
2. On the API Management services page, select your API Management instance.



1. Review the properties of your service on the Overview page.



**Migrate Amazon RDS for MySQL to Azure Database for MySQL using Data-in Replication:**

You can use methods such as MySQL dump and restore, MySQL Workbench Export and Import, or Azure Database Migration Service to migrate your MySQL databases to Azure Database for MySQL. You can migrate your workloads with minimum downtime by using a combination of open-source tools such as mysqldump or mydumper and myloader with Data-in Replication.

Data-in Replication is a technique that replicates data changes from the source server to the destination server based on the binary log file position method. In this scenario, the MySQL instance operating as the source (on which the database changes originate) writes updates and changes as events to the binary log.

## **Prerequisites:**

* Install the mysqlclient on your client computer to create a dump, and perform a restore operation on your target Azure Database for MySQL server.
* For larger databases, install mydumper and myloader for parallel dumping and restoring of databases.
* Create an instance of Azure Database for MySQL server that runs version 5.7 or 8.0.
* For compatibility reasons, have the source and target database servers on the same MySQL version.
* Have a primary key in each table. A lack of primary keys on tables can slow the replication process.
* Verify that all your tables use InnoDB. The Azure Database for MySQL server only supports the InnoDB storage engine.
* Verify that the target Azure Database for the MySQL server can connect to the source Amazon RDS for the MySQL server over port 3306.
* Ensure that the source Amazon RDS for the MySQL server allows both inbound and outbound traffic on port 3306.
* Make sure you provide **site-to-site connectivity** to your source server by using either **Azure ExpressRoute** or **Azure VPN Gateway**. For more information about creating a virtual network, see the Azure Virtual Network documentation. Also, see the quickstart articles with step-by-step details.
* Configure your source database server's network security groups to allow the target Azure Database for MySQL's server IP address.

## **Configure the target instance of Azure Database for MySQL:**

To configure the target instance of Azure Database for MySQL, which is the target for Data-in Replication:

1. Set the max\_allowed\_packet parameter value to the maximum of 1073741824, which is 1 GB. This value prevents any overflow issues related to long rows.
2. Set the slow\_query\_log, general\_log, audit\_log\_enabled, and query\_store\_capture\_mode parameters to OFF during the migration to help eliminate any overhead related to query logging.
3. Scale up the compute size of the target Azure Database for MySQL server to the maximum of 64 vCores. This size provides more compute resources when restoring the source server's database dump.

You can always scale back the compute to meet your application demands after the migration is complete.

1. Scale up the storage size to get more IOPS during the migration or increase the maximum IOPS for the migration.

## **Configure the source Amazon RDS for MySQL server:**

1. Confirm that binary logging is enabled on the source Amazon RDS for MySQL server. Check that automated backups are enabled, or ensure a read replica exists for the source Amazon RDS for MySQL server.
2. Ensure that the binary log files on the source server are retained until after the changes are applied on the target instance of Azure Database for MySQL.

With Data-in Replication, Azure Database for MySQL doesn't manage the replication process.

1. To check the binary log retention on the source Amazon RDS server to determine the number of hours the binary logs are retained, call the mysql.rds\_show\_configuration stored procedure:

mysql> call mysql.rds\_show\_configuration;

+------------------------+-------+-----------------------------------------------------------------------------------------------------------+

| name | value | description |

+------------------------+-------+-----------------------------------------------------------------------------------------------------------+

| binlog retention hours | 24 | binlog retention hours specifies the duration in hours before binary logs are automatically deleted. |

| source delay | 0 | source delay specifies replication delay in seconds between current instance and its master. |

| target delay | 0 | target delay specifies replication delay in seconds between current instance and its future read-replica. |

+------------------------+------- +-----------------------------------------------------------------------------------------------------------+

3 rows in set (0.00 sec)

1. To configure the binary log retention period, run the rds\_set\_configuration stored procedure to ensure that the binary logs are retained on the source server for the desired time. For example:

Mysql> Call mysql.rds\_set\_configuration(‘binlog retention hours', 96);

If you're creating a dump and restoring, the preceding command helps you catch up with the delta changes quickly.

There are two ways to capture a dump of data from the source Amazon RDS for MySQL server. One approach involves capturing a dump of data directly from the source server. The other approach involves capturing a dump from an Amazon RDS for MySQL read replica.

To capture a dump of data directly from the source server:

1. Ensure that you stop the writes from the application for a few minutes to get a transactionally consistent dump of data.
2. You can also temporarily set the read\_only parameter to a value of 1 so that writes aren't processed when you're capturing a dump of data.
3. After you stop the writes on the source server, collect the binary log file name and offset by running the command Mysql> Show master status;.
4. Save these values to start replication from your Azure Database for MySQL server.
5. To create a dump of the data, execute mysqldump by running the following command:

$ mysqldump -h hostname -u username -p –single-transaction –databases dbnames –order-by-primary> dumpname.sql

If stopping writes on the source server isn't an option or the performance of dumping data isn't acceptable on the source server, capture a dump on a replica server:

1. Create an Amazon MySQL read replica with the same configuration as the source server. Then create the dump there.
2. Let the Amazon RDS for MySQL read replica catch up with the source Amazon RDS for MySQL server.
3. When the replica lag reaches 0 on the read replica, stop replication by calling the stored procedure mysql.rds\_stop\_replication.

Mysql> call mysql.rds\_stop\_replication;

1. With replication stopped, connect to the replica. Then run the SHOW SLAVE STATUS command to retrieve the current binary log file name from the Relay\_Master\_Log\_File field and the log file position from the Exec\_Master\_Log\_Pos field.
2. Save these values to start replication from your Azure Database for MySQL server.
3. To create a dump of the data from the Amazon RDS for MySQL read replica, execute mysqldump by running the following command:

$ mysqldump -h hostname -u username -p –single-transaction –databases dbnames –order-by-primary> dumpname.sql

## **Link source and replica servers to start Data-in Replication**

1. To restore the database by using mysql native restore, run the following command:

$ mysql -h <target\_server> -u <targetuser> -p < dumpname.sql

1. Sign in to the source Amazon RDS for MySQL server, and set up a replication user. Then grant the necessary privileges to this user.

If you're using SSL, run the following commands:

Mysql> CREATE USER 'syncuser'@'%' IDENTIFIED BY 'userpassword';

Mysql> GRANT REPLICATION SLAVE, REPLICATION CLIENT on \*.\* to 'syncuser'@'%' REQUIRE SSL;

Mysql> SHOW GRANTS FOR syncuser@'%';

If you're not using SSL, run the following commands:

Mysql> CREATE USER 'syncuser'@'%' IDENTIFIED BY 'userpassword';

Mysql> GRANT REPLICATION SLAVE, REPLICATION CLIENT on \*.\* to 'syncuser'@'%';

Mysql> SHOW GRANTS FOR syncuser@'%';

1. To link the Amazon RDS for MySQL source server and the Azure Database for MySQL target server, sign in to the target Azure Database for MySQL server. Set the Amazon RDS for MySQL server as the source server by running the following command:

CALL mysql.az\_replication\_change\_master('source\_server','replication\_user\_name','replication\_user\_password',3306,'<master\_bin\_log\_file>',master\_bin\_log\_position,'<master\_ssl\_ca>');

1. To start replication between the source Amazon RDS for MySQL server and the target Azure Database for MySQL server, run the following command:

Mysql> CALL mysql.az\_replication\_start;

1. To check the status of the replication on the replica server, run the following command:

Mysql> show slave status\G

**Migrate Amazon RDS for MySQL to Azure Database for MySQL using MySQL Workbench**

## **Prerequisites:**

* Make sure the Character set of the source and target databases are the same.
* Set the wait timeout to a reasonable time depending on the amount data or workload you want to import or migrate.
* Set the max\_allowed\_packet parameter to a reasonable amount depending on the size of the database you want to import or migrate.
* Verify that all of your tables use InnoDB, as Azure Database for MySQL Server only supports the InnoDB Storage engine.
* Remove, replace, or modify all triggers, stored procedures, and other functions containing root user or super user definers (Azure Database for MySQL doesn’t support the Super user privilege). To replace the definers with the name of the admin user that is running the import process, run the following command:

DELIMITER; ;/\*!50003 CREATE\*/ /\*!50017 DEFINER=`root`@`127.0.0.1`\*/ /\*!50003

DELIMITER;

/\* Modified to \*/

DELIMITER;

/\*!50003 CREATE\*//\*!50017 DEFINER=`AdminUserName`@`ServerName`\*/ /\*!50003

DELIMITER;

* If User Defined Functions (UDFs) are running on your database server, you need to delete the privilege for the mysql database. To determine if any UDFs are running on your server, use the following query:

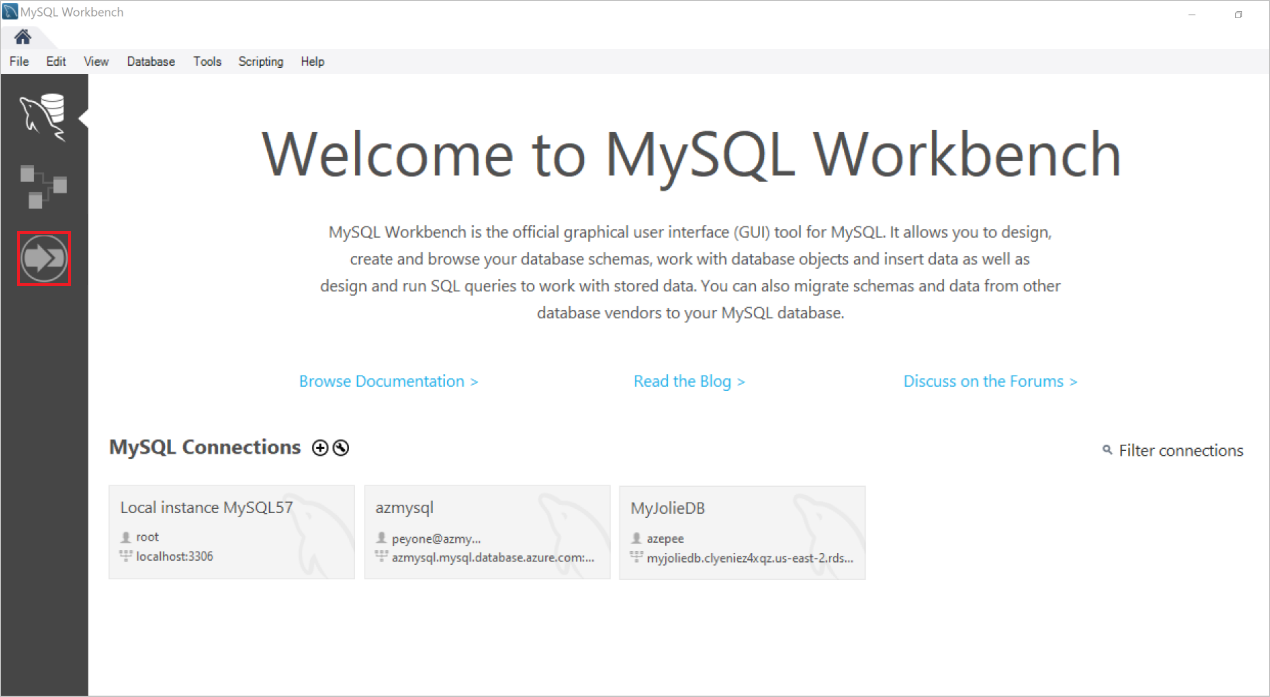
SELECT \* FROM mysql.func;

* If you discover that UDFs are running, you can drop the UDFs by using the following query:

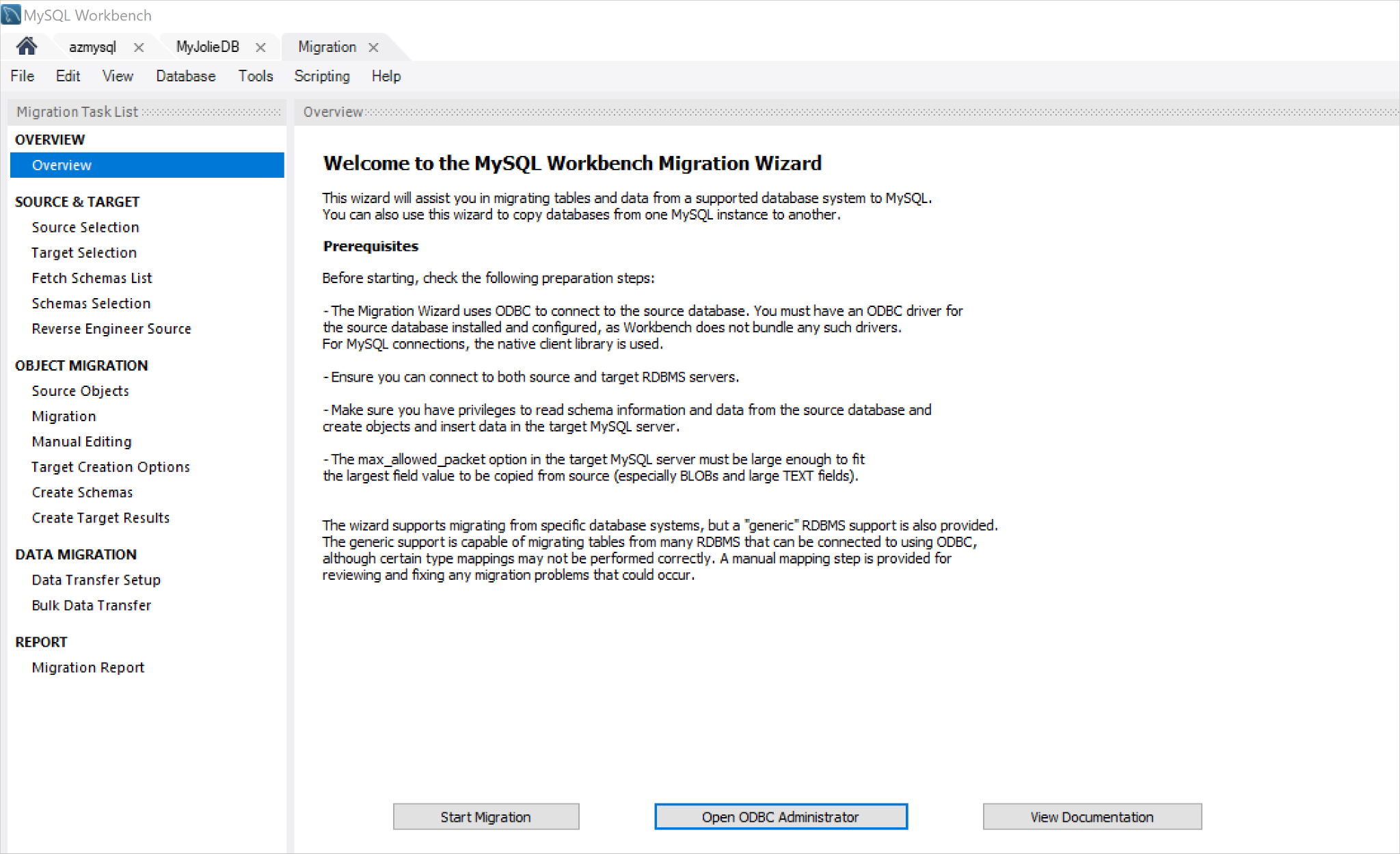
DROP FUNCTION your\_UDFunction;

## **Begin the migration process:**

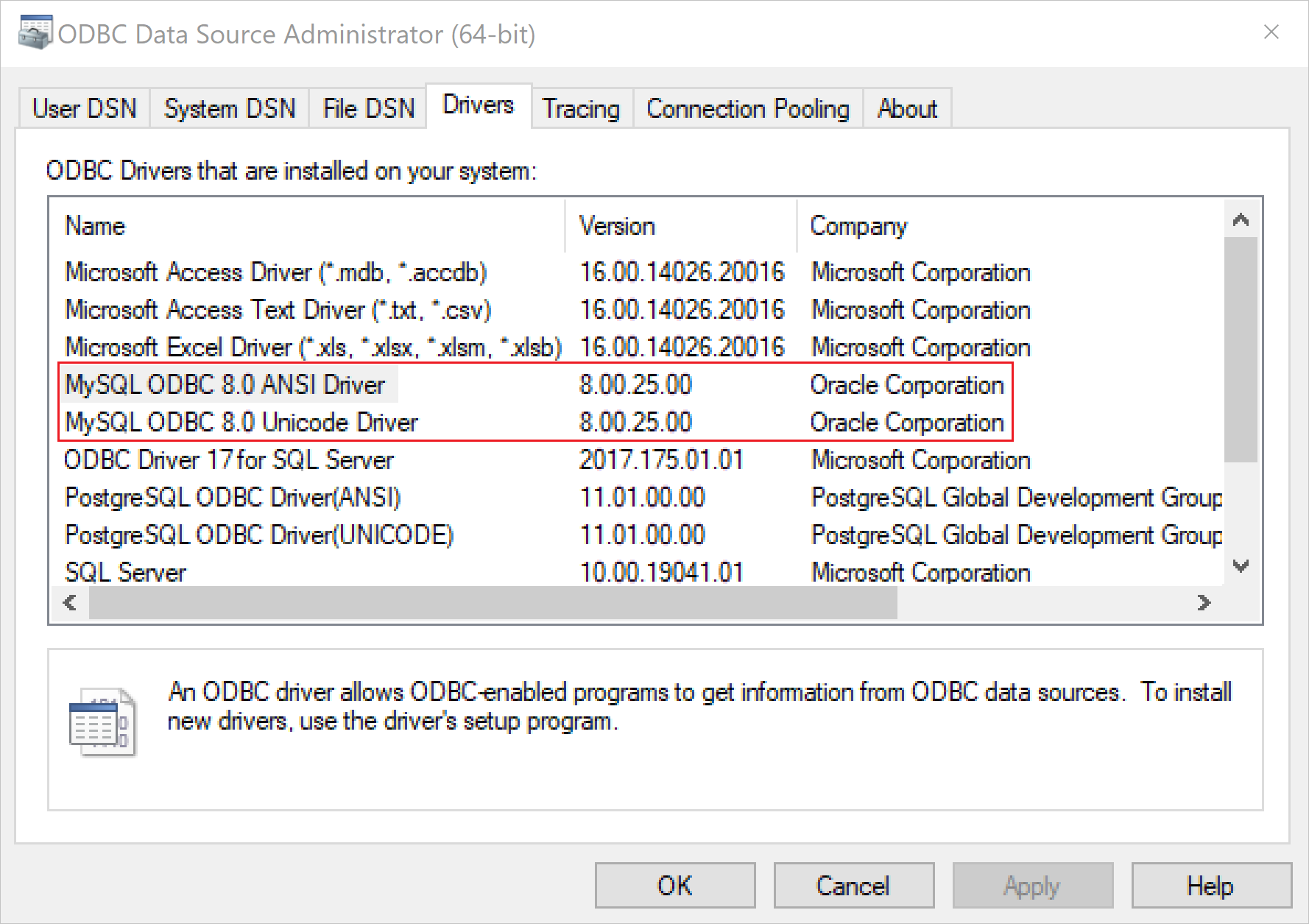
1. To start the migration process, sign in to MySQL Workbench, and then select the home icon.
2. In the left-hand navigation bar, select the Migration Wizard icon, as shown in the screenshot below.



1. The Overview page of the Migration Wizard is displayed, as shown below.

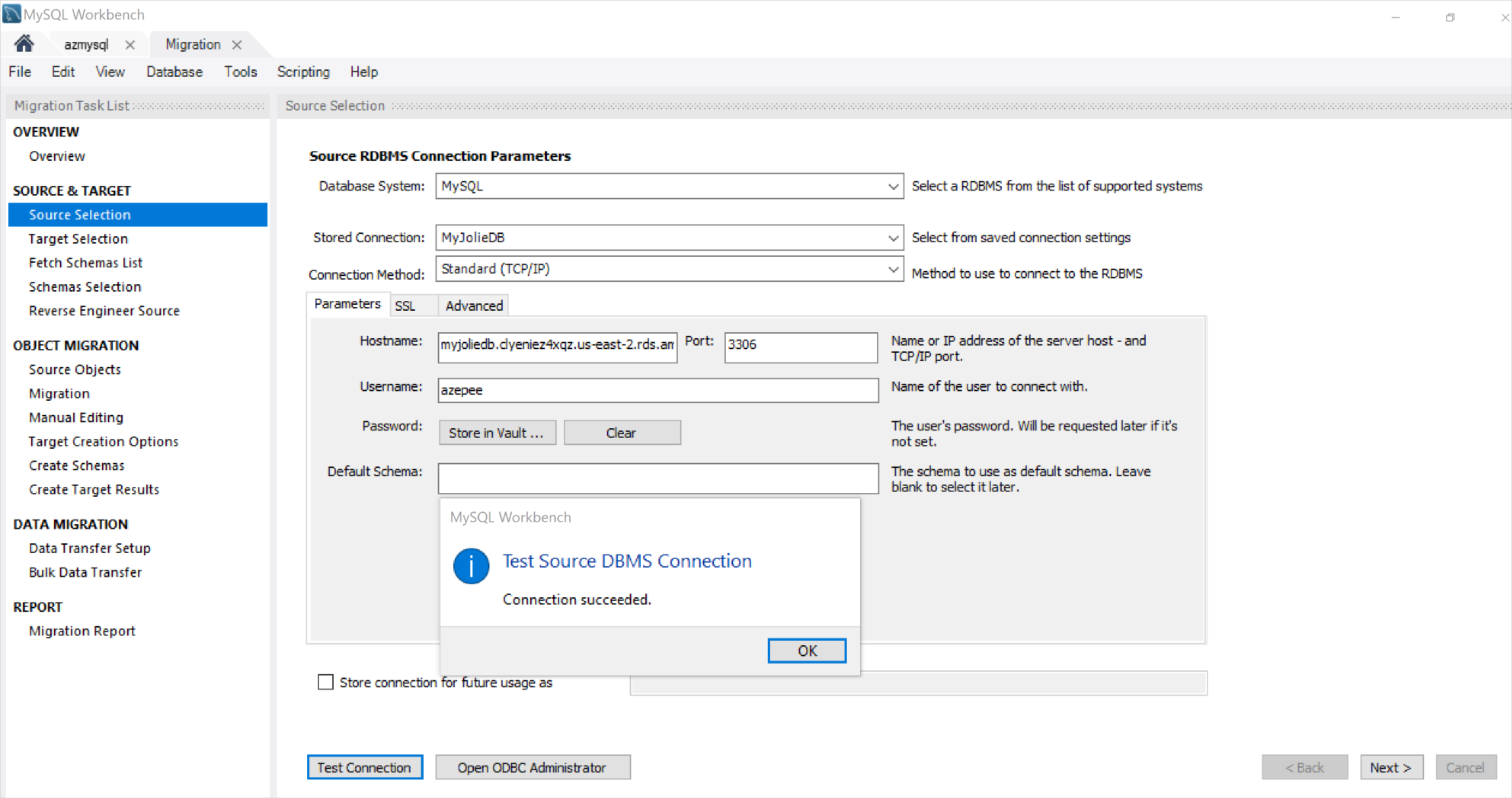


1. Determine if you have an ODBC driver for MySQL Server installed by selecting Open ODBC Administrator.



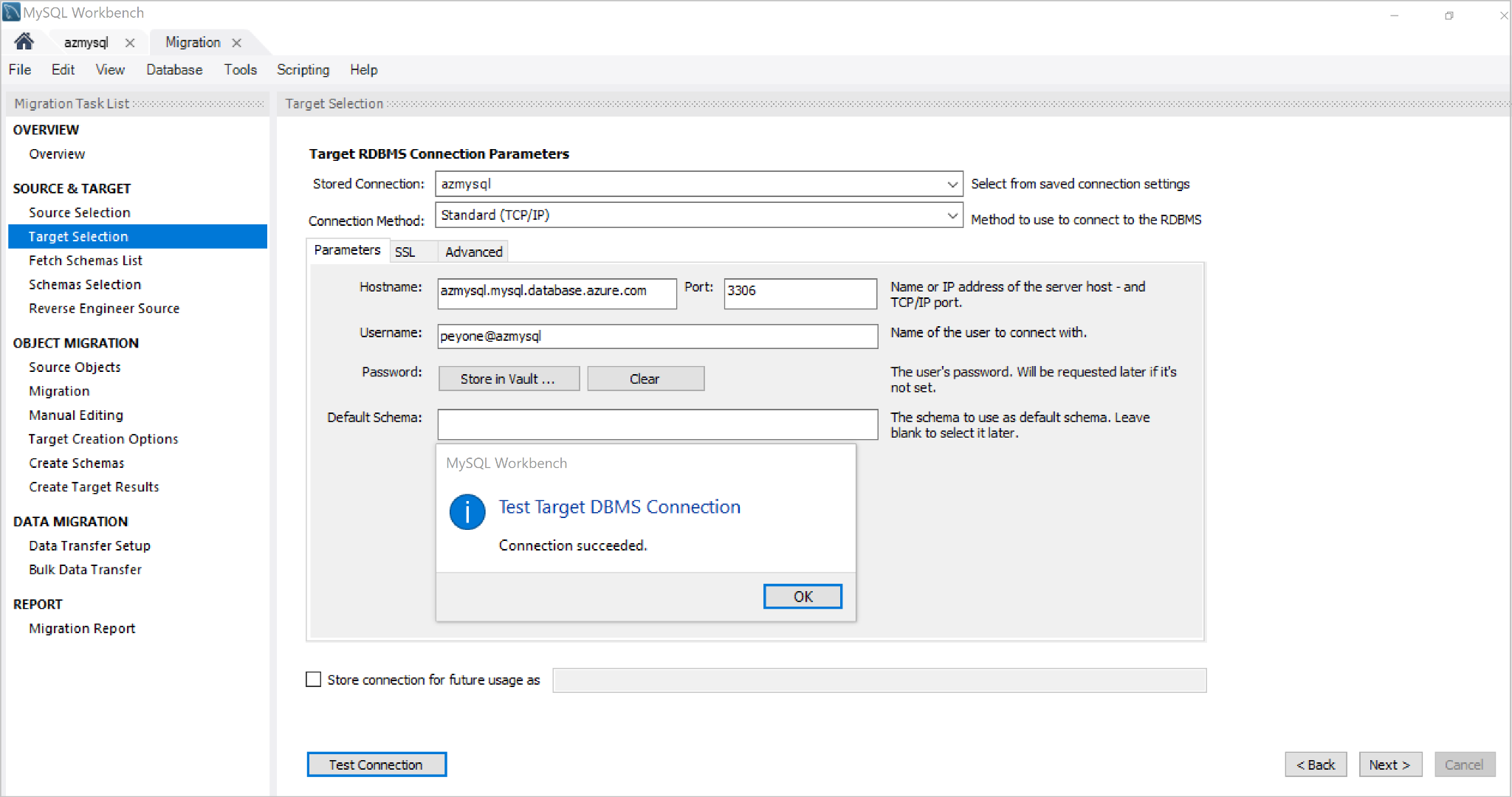
## **Configure source database server connection parameters**

1. On the Overview page, select Start Migration.
2. In the Stored Connection field, select one of the saved connection settings for that RDBMS.
3. You can save connections by marking the checkbox at the bottom of the page and providing a name of your preference.
4. In the Connection Method field, select Standard TCP/IP.
5. In the Hostname field, specify the name of your source database server.
6. In the Port field, specify 3306, and then enter the username and password for connecting to the server.
7. In the Database field, enter the name of the database you want to migrate if you know it; otherwise leave this field blank.
8. Select Test Connection to check the connection to your MySQL Server instance.
9. If you’ve entered the correct parameters, a message appears indicating a successful connection attempt.



## **Configure target database server connection parameters**

1. On the Target Selection page, set the parameters to connect to your target MySQL Server instance using a process similar to that for setting up the connection to the source server.
2. To verify a successful connection, select Test Connection.



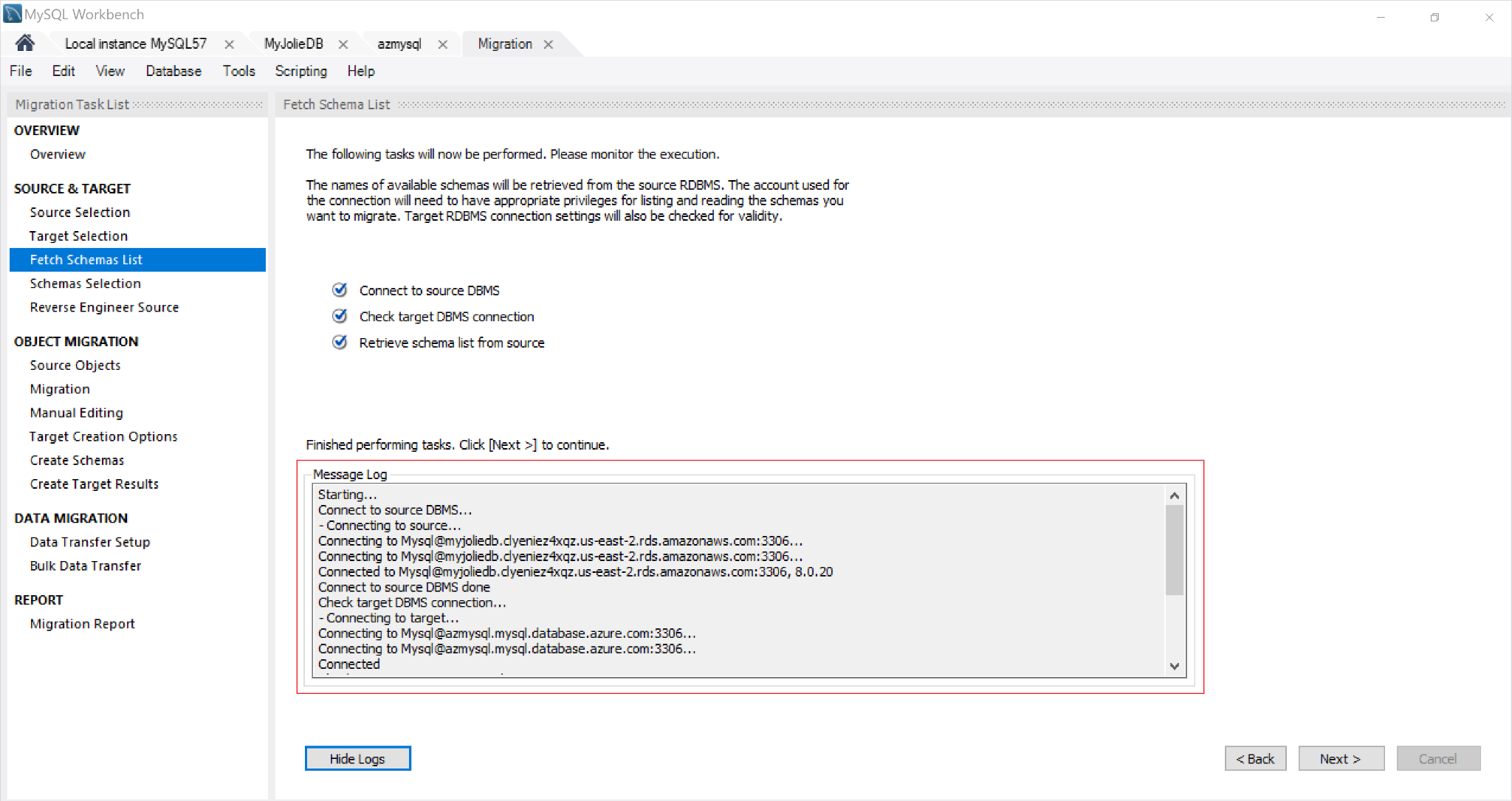
1. Select **Next**

## **Select the schemas to migrate**

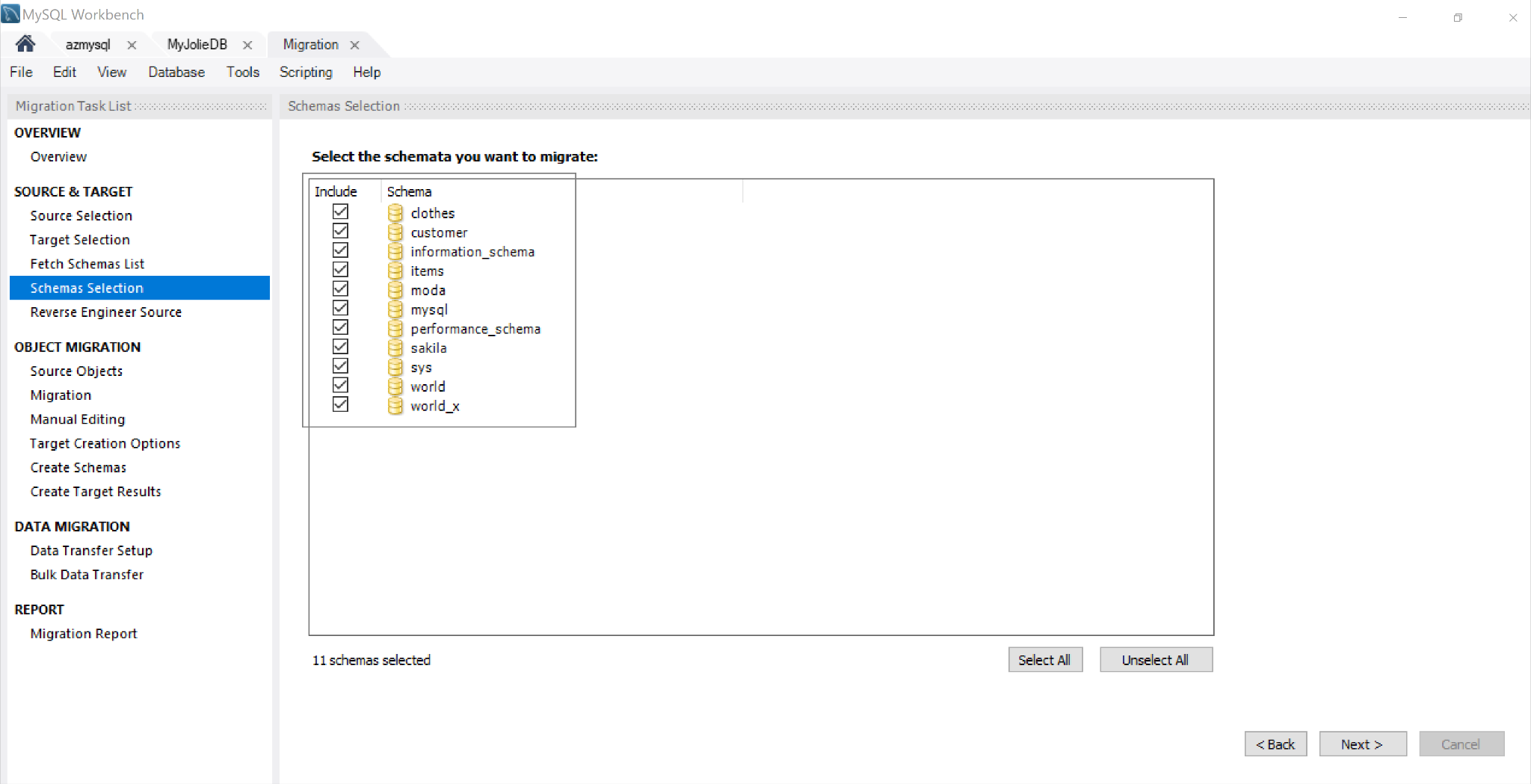
The Migration Wizard will communicate to your MySQL Server instance and fetch a list of schemas from the source server.

1. Select Show logs to view this operation.

The screenshot below shows how the schemas are being retrieved from the source database server.



1. Select Next to verify that all the schemas were successfully fetched.



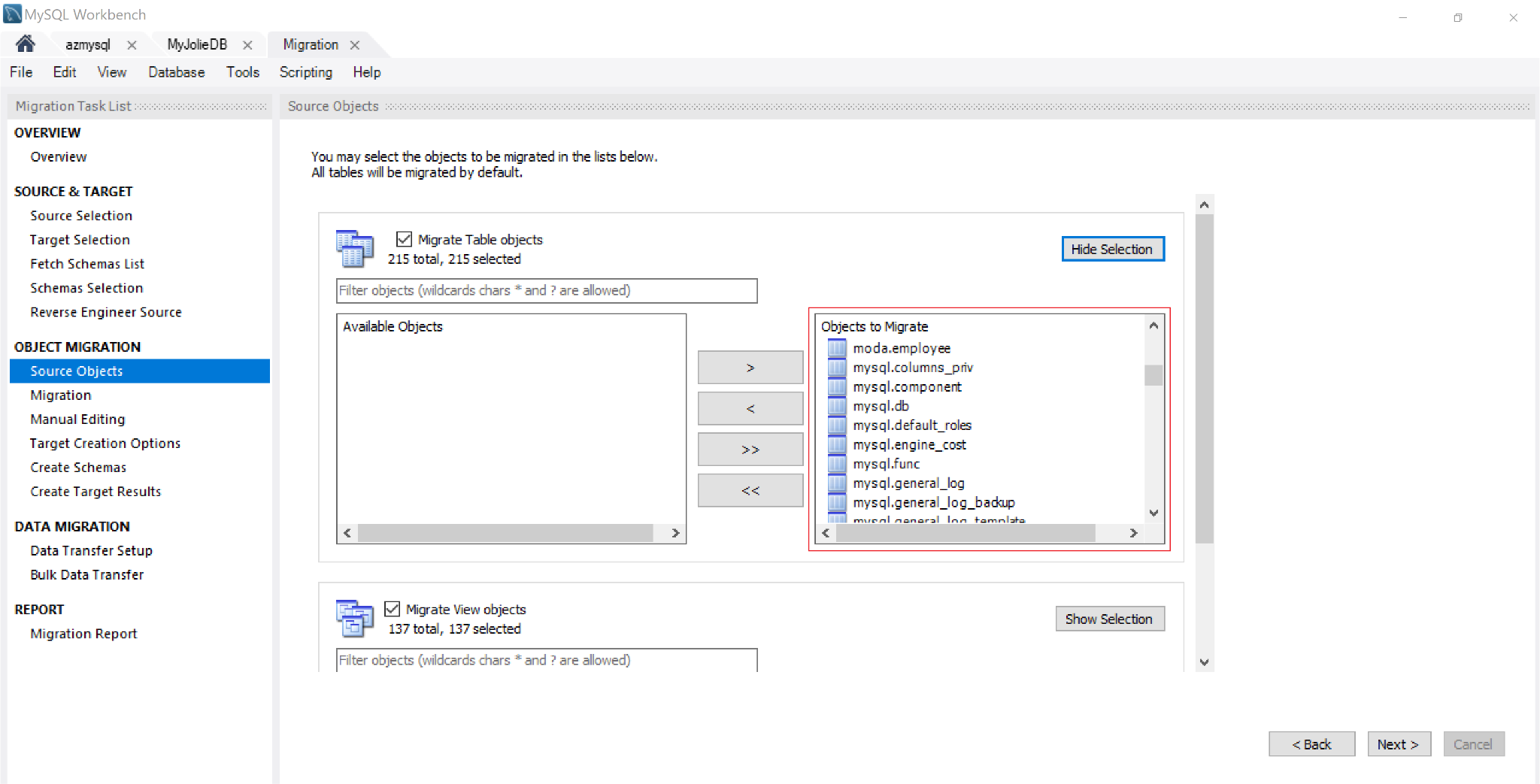
You can only migrate schemas that appear in this list.

1. Select the schemas that you want to migrate, and then select Next.

## **Object migration:**

1. Select Show Selection, and then, under Available Objects, select and add the objects that you want to migrate.

When you've added the objects, they'll appear under Objects to Migrate, as shown in the screenshot below.



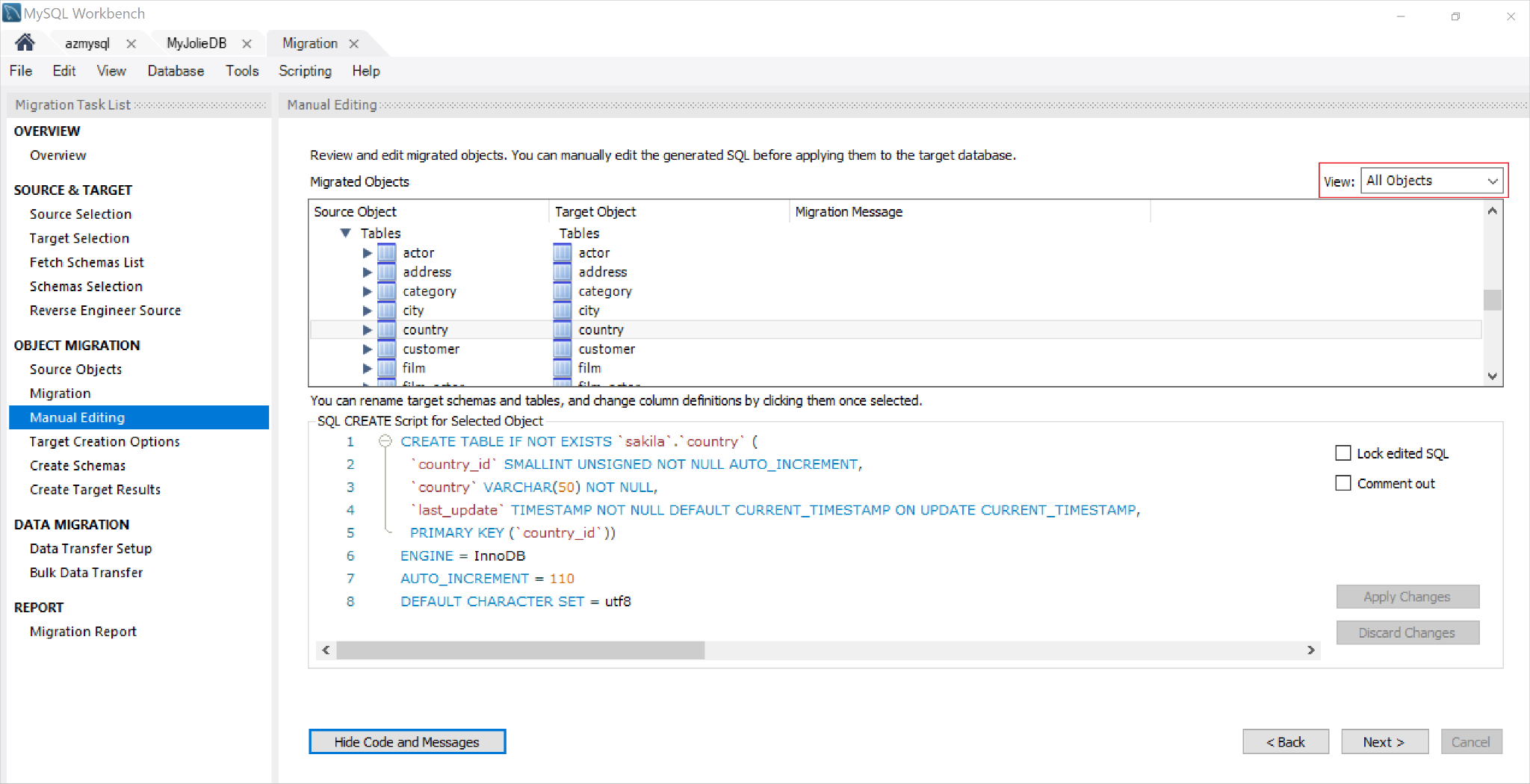
In this scenario, we’ve selected all table objects.

1. Select **Next**.

## **Edit data:**

In this section, you have the option of editing the objects that you want to migrate.

1. On the Manual Editing page, notice the View drop-down menu in the top-right corner.

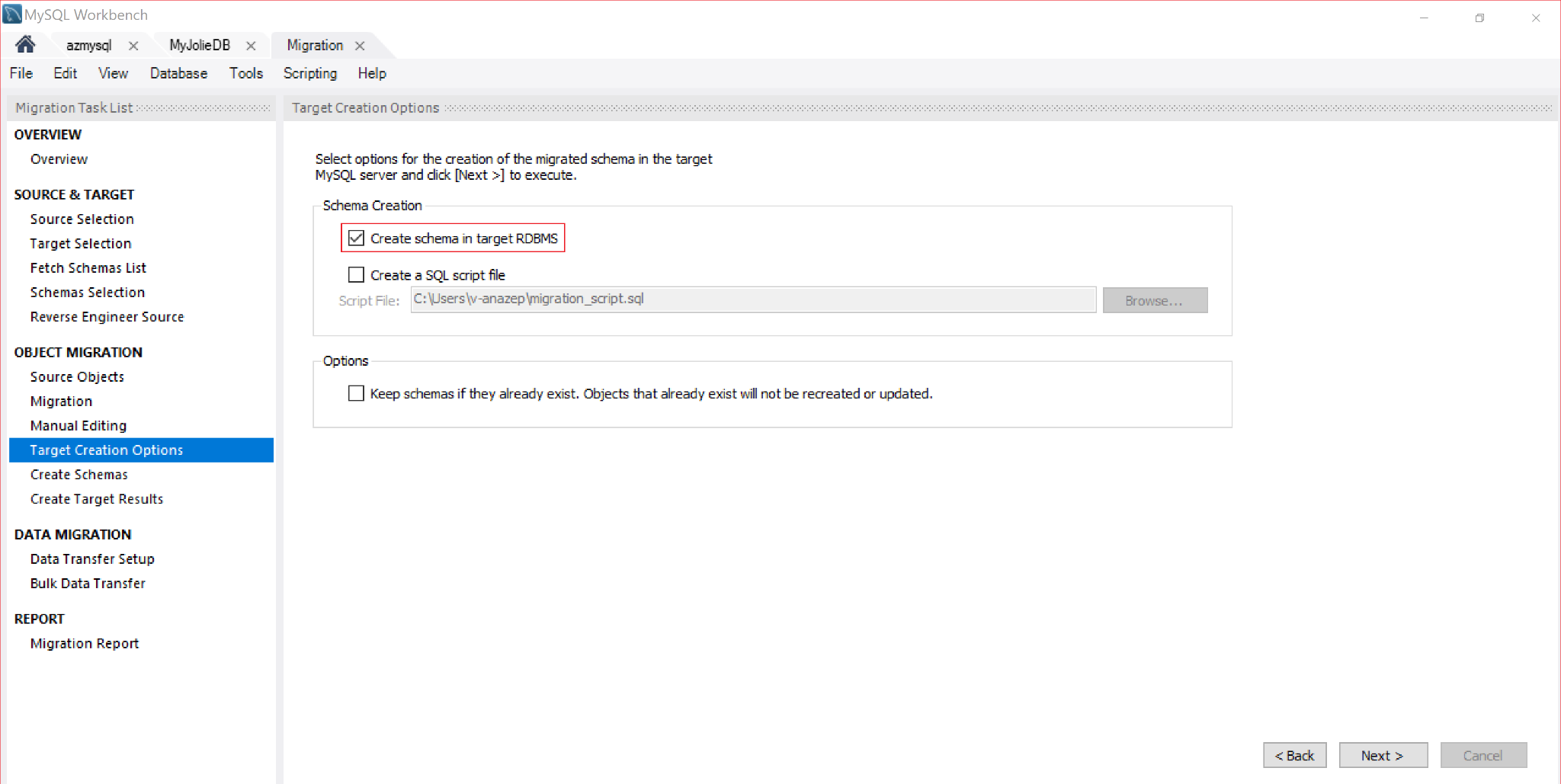


1. Select **Next**

## **Create the target database:**

1. Select the Create schema in target RDBMS check box.

You can also choose to keep already existing schemas, so they won't be modified or updated.

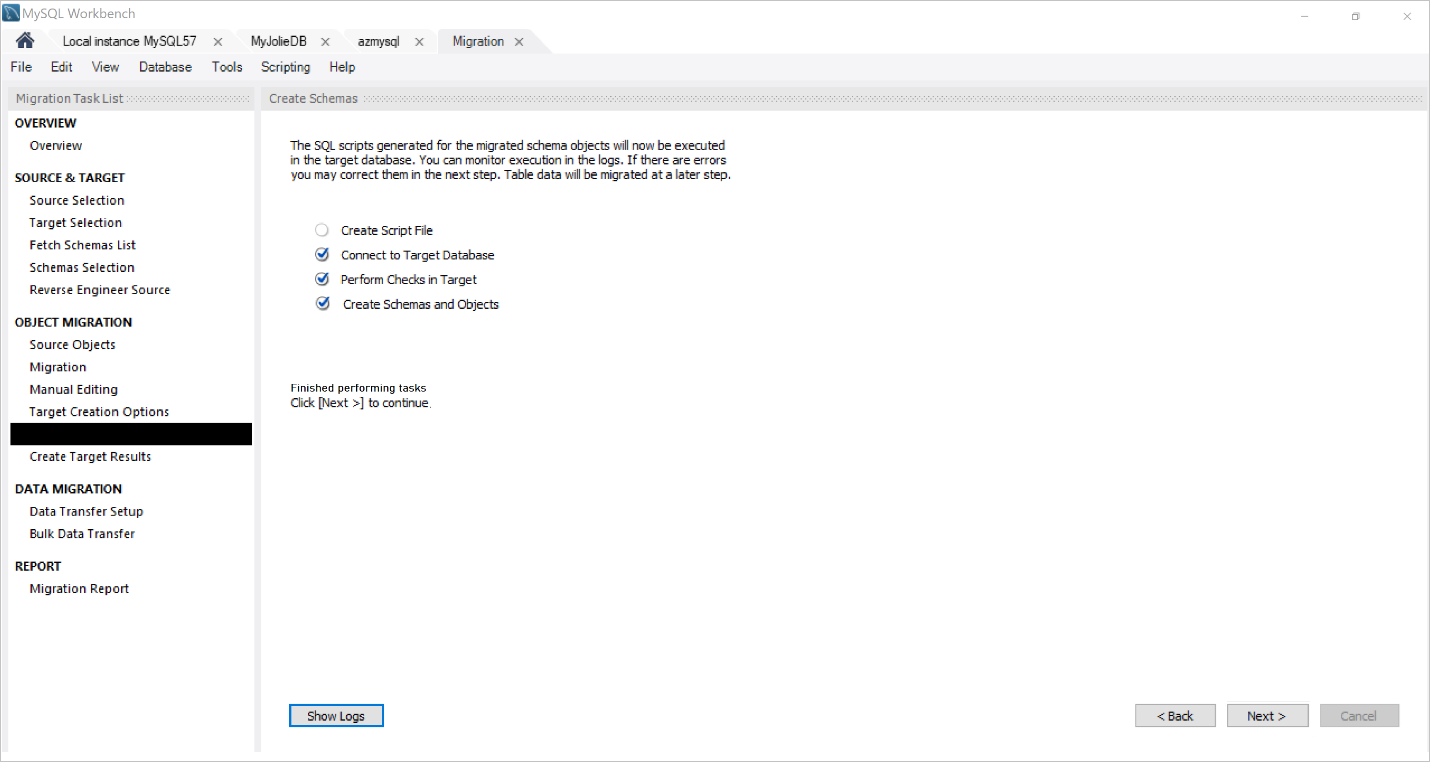


In this article, we've chosen to create the schema in target RDBMS, but you can also select the Create a SQL script file check box to save the file on your local computer or for other purposes.

1. Select **Next.**

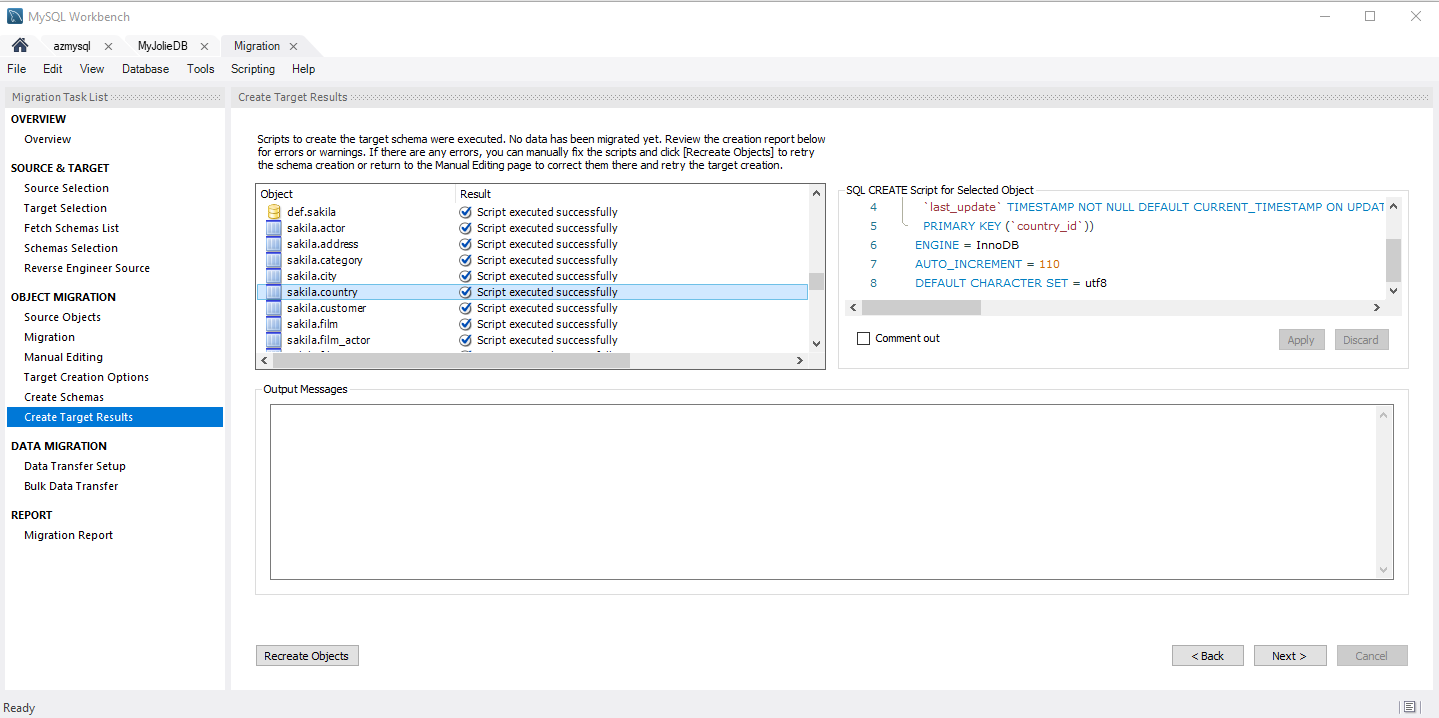
## **Run the MySQL script to create the database objects:**

Since we've elected to create schema in the target RDBMS, the migrated SQL script will be executed in the target MySQL server. You can view its progress as shown in the screenshot below:



1. After the creation of the schemas and their objects completes, select Next.

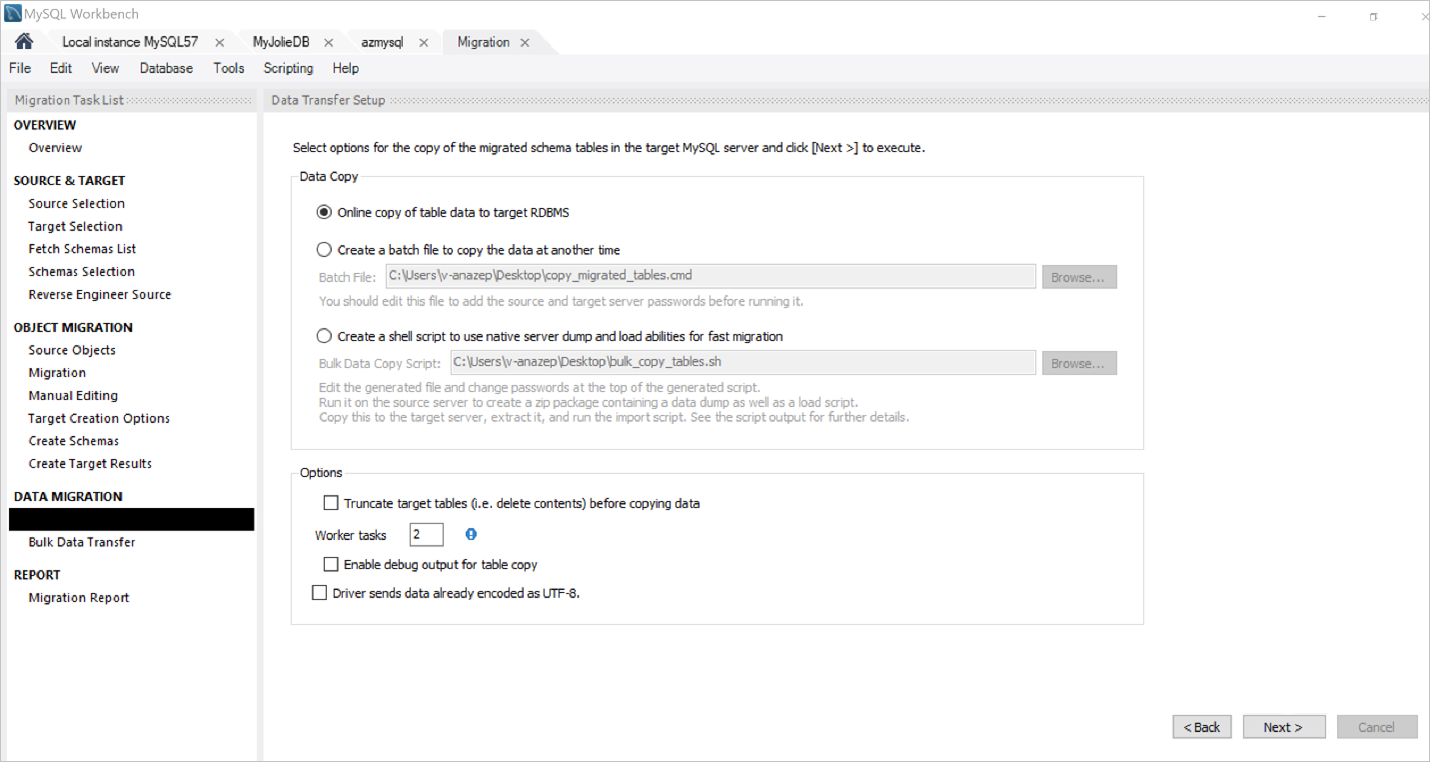
On the Create Target Results page, you’re presented with a list of the objects created and notification of any errors that were encountered while creating them, as shown in the following screenshot.



1. Review the detail on this page to verify that everything completed as intended.
2. For this article, we don’t have any errors. If there's no need to address any error messages, you can edit the migration script.
3. In the Object box, select the object that you want to edit.
4. Under SQL CREATE script for selected object, modify your SQL script, and then select Apply to save the changes.
5. Select Recreate Objects to run the script including your changes.
6. Select **Next**

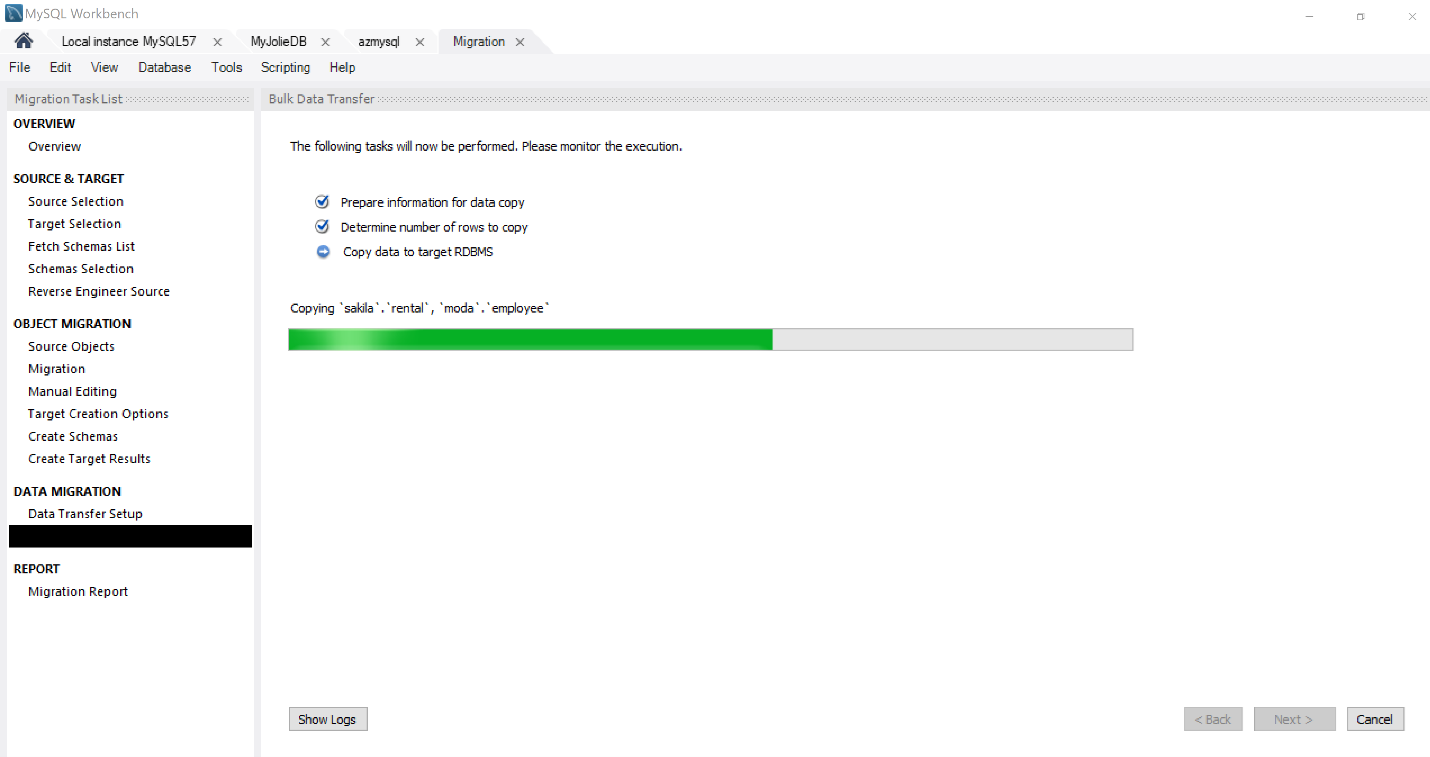
## **Transfer data:**

This part of the process moves data from the source MySQL Server database instance into your newly created target MySQL database instance. Use the Data Transfer Setup page to configure this process.



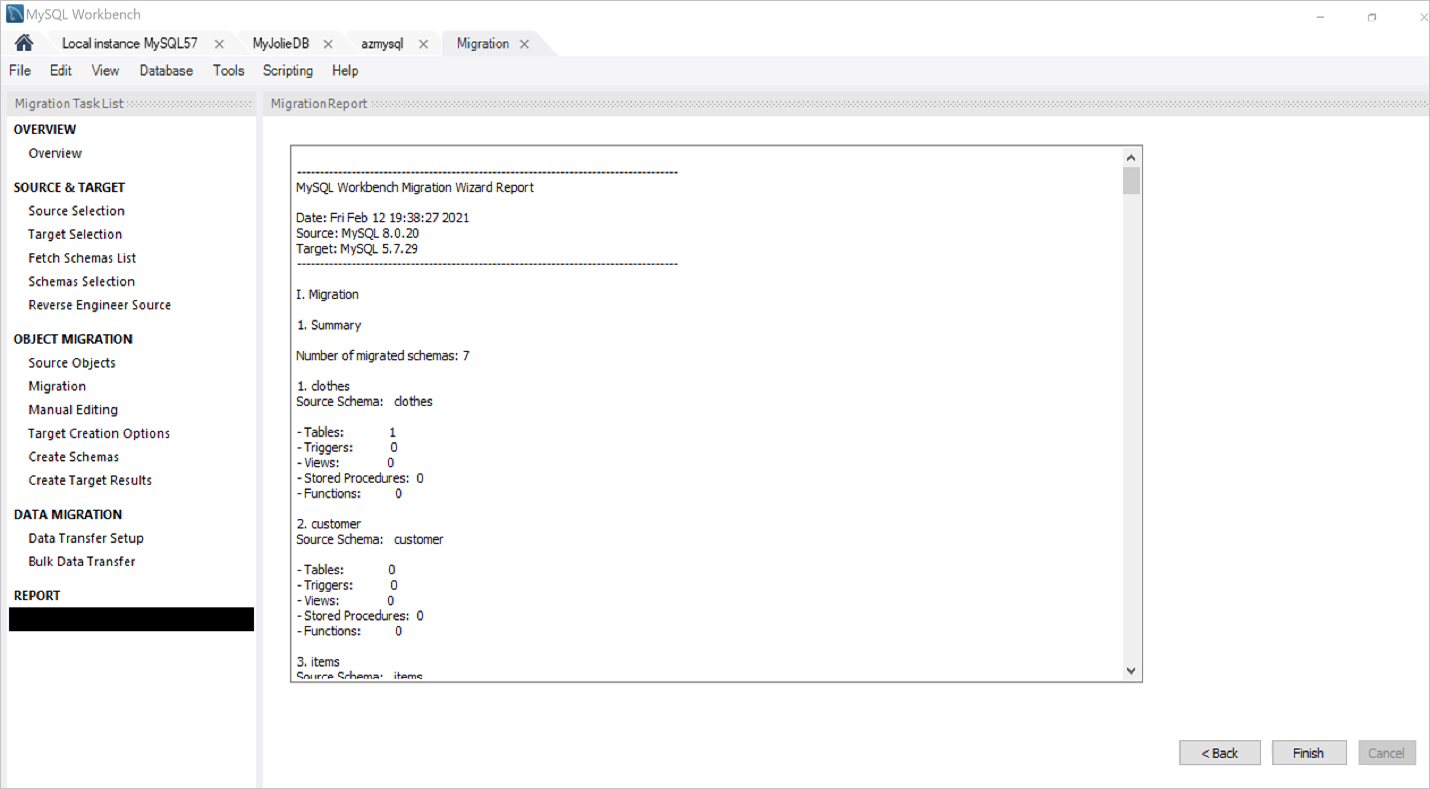
1. To begin the actual process of transferring data, select Next.

The progress of the data transfer process appears as shown in the following screenshot.



1. After the transfer completes, select Next.

The Migration Report page appears, providing a report summarizing the whole process, as shown on the screenshot below:



1. Select Finish to close the Migration Wizard.