Deploying to Azure with Kubernetes

This page shows how to deploy a complete Metaflow stack powered by Kubernetes on Microsoft Azure. For more information about the deployment, see deployment details, advanced options and FAQ.

1. Preparation

Terraform Tooling

Terraform is a popular infrastructure-as-code tool for managing cloud resources. We have published a set of Terraform templates here for setting up Metaflow on Microsoft Azure. Terraform needs to be installed on your system in order to use these templates.

1. Install Terraform by following these instructions.
2. Download Metaflow on Azure terraform templates:

```
git clone git@github.com:outerbounds/metaflow-tools.git
```

Azure Command Line Interface

This is the official CLI tool ("az") published by Microsoft for working with Azure. It will be used by Terraform when applying our templates (e.g. for authentication with Azure). Please install it by following these instructions.

kubectl Command Line Interface

kubectl is a standard CLI tool for working with Kubernetes clusters. It will be used by Terraform when applying our templates (e.g. for deploying some services to your Azure Kubernetes Service cluster). Please install it by following these instructions.

2. Provision Azure Resources

See here for the exact set of resources to be provisioned. Also, note the permissions that are needed.

Login to Azure

You must be logged onto Azure as an account with sufficient permissions to provision the required resources. Use the Azure CLI (az):

```
az login
```

Initialize your Terraform Workspace

From your metaflow-tools/azure/terraform directory, run:

```
terraform init
```

Set org_prefix

Create a TF vars file FILE.tfvars (FILE could be something else), with this content.

```
org_prefix = "yourorg"  # use something short and distinctive
```

Some Azure resources must have globally unique names:

- Azure storage account
- Azure PostgreSQL Flexible Server

org_prefix will be used to ensure this uniqueness.

Uncomment this line, and set org_prefix to a value of your choice. Short and distinctive is best.

Optional: Enable Argo Events

To enable event triggering for Metaflow, add the following line in FILE.tfvars:

```
enable_argo=true
```

For more technical context, see this page about event triggering.

Optional: Enable Airflow

Optionally, you can include Apache Airflow as the production orchestrator for Metaflow in your deployment by including the following lines in FILE.tfvars:

```
deploy_airflow=true
```

Setting deploy_airflow=true will create a storage blob-container named airflow-logs, provide blob-container read and write permissions to the service principal and deploy Airflow in the AKS cluster with a LocalExecutor. The Airflow installation will store the logs in the airflow-logs blob container.

Apply Terraform Template to Provision Azure Infrastructure

From your local metaflow-tools/azure/terraform directory, run:

```
terraform apply -target="module.infra" -var-file="FILE.tfvars"
```

A plan of action will be printed to the terminal. You should review it before accepting.
See details for what to expect.

Common Resource Provisioning Hiccups

PostgreSQL Provisioning API Errors (on Azure Side)

If you do not create Azure PostgreSQL Flexible Server instances often, Azure API may be flaky for you initially:

```
| Error: waiting for creation of the Postgresql Flexible Server "metaflow-database-server-xyz" (Resource Group "rg-db-metaflow-xyz"):
| Code="InternalServerError" Message="An unexpected error occured while processing the request. Tracking ID: 'xyz'"
|
|   with module.infra.azurerm_postgresql_flexible_server.metaflow_database_server,
|   on infra/database.tf line 20, in resource "azurerm_postgresql_flexible_server" "metaflow_database_server":
|   20: resource "azurerm_postgresql_flexible_server" "metaflow_database_server" {
```

In our experience, waiting 20 mins and trying again resolves this issue. This appears to be a one-time phenomenon - future stack spin-ups do not encounter such InternalServerErrors.

Node Pool Provisioning

We have hard-coded some default instance types to be used for Kubernetes nodes as well as worker pools (taskworkers). Depending on the real-time availability of such instances in your region or availability zone, you may consider choosing alternate instance types.

VM Availability issues might look something like this:

```
| Error: waiting for creation of Node Pool: (Agent Pool Name "taskworkers" / Managed Cluster Name
"metaflow-kubernetes-xyz" /
| Resource Group "rg-k8s-metaflow-xyz"): Code="ReconcileVMSSAgentPoolFailed"
Message="Code=\"AllocationFailed\" Message=\"Allocation failed.
| We do not have sufficient capacity for the requested VM size in this region. Read more about improving
likelihood of allocation success
| at http://aka.ms/allocation-guidance\""
```

VM quotas may also cause provisioning to fail - we recommend working with your Azure admin to raise quotas, and/or pick other instance types:

```
| Error: creating Node Pool: (Agent Pool Name "taskworkers" / Managed Cluster Name "metaflow-
kubernetes-default" / Resource Group "rg-k8s-metaflow-default"):
| containerservice.AgentPoolsClient#CreateOrUpdate: Failure sending request: StatusCode=400 -- Original
Error: Code="PreconditionFailed"
| Message="Provisioning of resource(s) for Agent Pool taskworkers failed. Error: {\n  \"code\":
\"InvalidTemplateDeployment\",\n
| \"message\": \"The template deployment '8b1a99f1-e35e-44be-a8ac-0f82009b7149' is not valid according
to the validation procedure.
| The tracking id is 'xyz'. See inner errors for details.\",\n  \"details\":
| [\n  {\n  \"code\": \"QuotaExceeded\",\n  \"message\": \"Operation could not be completed as it results
in exceeding approved standardDv5Family Cores quota.
| Additional details - Deployment Model: Resource Manager, Location: westeurope, Current Limit: 0, Current
Usage: 0,
| Additional Required: 4, (Minimum) New Limit Required: 4.
| Submit a request for Quota increase at https://<AZURE_LINK> by specifying parameters listed in the
'Details' section for deployment to succeed.
| Please read more about quota limits at https://docs.microsoft.com/en-us/azure/azure-supportability/per-
vm-quota-requests\"\n  }\n  ]\n }"
```

3. Deploy Metaflow Services to AKS Cluster

Apply Terraform Template to Deploy Services

From your local metaflow-tools/azure/terraform directory, run:

terraform apply -target="module.services" -var-file="FILE.tfvars"

4. End User Setup Instructions

When the command above completes, it will print a set of setup instructions for Metaflow end users (folks who will be writing and running flows). These instructions are meant to get end users started on running flows quickly.

You can access the Terraform instruction output at any time by running (from metaflow-tools/azure/terraform directory):

terraform output -raw END_USER_SETUP_INSTRUCTIONS

Sample Output

Setup instructions for END USERS (e.g. someone running Flows vs the new stack):

There are three steps:
1. Ensuring Azure access
2. Configure Metaflow
3. Run port forwards
4. Install necessary Azure Python SDK libraries

STEP 1: Ensure you have sufficient access to these Azure resources on your local workstation:

- AKS cluster ("aks-ob-metaflow-minion") ("Azure Kubernetes Service Contributor" + "Azure Kubernetes Service Cluster User Role")
- Azure Storage ("metaflow-storage-container" in the storage account "stobmetaflowminion") ("Storage Blob Data Contributor")

You can use "az login" as a sufficiently capabable account. To see the credentials for the service principal (created by terraform) that is capable, run this:

$ terraform output -raw SERVICE_PRINCIPAL_CREDENTIALS

Use the credentials with "az login"

$ az login --service-principal -u $AZURE_CLIENT_ID -p $AZURE_CLIENT_SECRET --tenant $AZURE_TENANT_ID

Configure your local Kubernetes context to point to the the right Kubernetes cluster:

$ az aks get-credentials --resource-group rg-metaflow-minion-westus --name aks-ob-metaflow-minion

STEP 2: Configure Metaflow:

$ metaflow configure azure
$ metaflow configure kubernetes

Use these values when prompted:

METAFLOW_DATASTORE_SYSROOT_AZURE=metaflow-storage-container/tf-full-stack-sysroot
METAFLOW_AZURE_STORAGE_BLOB_SERVICE_ENDPOINT=https://stobmetaflowminion.blob.core.windows.net/
METAFLOW_KUBERNETES_SECRETS=metaflow-azure-storage-credentials
METAFLOW_SERVICE_URL=http://127.0.0.1:8080/
METAFLOW_SERVICE_INTERNAL_URL=http://metadata-service.default:8080/
[For Argo only] METAFLOW_KUBERNETES_NAMESPACE=argo

Note: you can skip METAFLOW_SERVICE_AUTH_KEY (leave it blank)

STEP 3: Setup port-forwards to services running on Kubernetes:

option 1 - run kubectl's manually:
$ kubectl port-forward deployment/metadata-service 8080:8080
$ kubectl port-forward deployment/metaflow-ui-backend-service 8083:8083
$ kubectl port-forward deployment/metaflow-ui-static-service 3000:3000
$ kubectl port-forward -n argo deployment/argo-server 2746:2746

option 2 - this script manages the same port-forwards for you (and prevents timeouts)

$ python metaflow-tools/scripts/forward_metaflow_ports.py [--include-argo]

STEP 4: Install Azure Python SDK
$ pip install azure-storage-blob azure-identity