Leveraging Confidential Computing on Kubernetes on Azure

A starter guide for developers

Version 2.0 (Draft), July 2020

For the latest information about Azure, please see
https://azure.microsoft.com/en-us/overview/

For the latest information on Azure Confidential Computing (ACC), please see
https://azure.microsoft.com/en-us/solutions/confidential-compute/

For the latest information about open source on Azure, please see
https://azure.microsoft.com/en-us/overview/choose-azure-opensource/

For the latest information about Kubernetes on Azure, please see
https://azure.microsoft.com/en-us/topic/what-is-kubernetes/

For the latest information on the Open Enclave (OE) SDK, please see
https://openenclave.io/sdk/

This page is intentionally left blank.

Table of contents

Notice

This guide in intended for developers to create a Kubernetes cluster on Azure using nodes having the Intel SGX technologies thanks to Azure Confidential Computing. As such, it also illustrates a way to build and execute so-called Trusted Execution Environment (TEE) based applications using the Microsoft Open Enclave SDK (OESDK) in in containers. The OESDK is available in open source at https://openenclave.io/sdk/.

MICROSOFT DISCLAIMS ALL WARRANTIES, EXPRESS, IMPLIED, OR STATUTORY, IN RELATION WITH THE INFORMATION CONTAINED IN THIS WHITE PAPER. The white paper is provided "AS IS" without warranty of any kind and is not to be construed as a commitment on the part of Microsoft.

Microsoft cannot guarantee the veracity of the information presented. The information in this guide, including but not limited to an internet website and URL references, is subject to change at any time without notice. Furthermore, the opinions expressed in this guide represent the current vision of Microsoft France on the issues cited at the date of publication of this guide and are subject to change at any time without notice.

Publication: July 2020
Version 2.0

About this guide

Welcome to the **Leveraging Confidential Computing on Kubernetes using Open Enclave SDK on Azure** guide for developers.

This document is part of a series of guides that covers Confidential Computing in the Cloud, and the Edge, and considerations that pertain to it from a development perspective and/or an infrastructure one. This series of guides is available at https://aka.ms/CCDevGuides.

The focus of this guide is to present a way to create a Kubernetes cluster on Azure with Confidential Computing. Confidential Computing adds new data security capabilities using Trusted Execution Environment[1] (TEE), a.k.a. enclaves, or encryption mechanisms to protect your data while in use.

As such, this guide is intended as practical approach to using TEE and TEE based application on a Kubernetes cluster, and thus won't expand on theorical concerns on what specifically a TEE is, and the guarantees it provides, etc.

For more theorical approaches of Confidential Computing, please refer to other guides of this series, and more particularly to the **Building and Executing Trusted Execution Environment (TEE) based applications on Azure – A starter guide for developers**.

For the sake of this guide, you will simply need to know that TEEs are hardware or software implementations that safeguard data being processed from access outside the TEE. The TEE type that will be illustrated in this guide is the SGX enclave as enabled by the Intel Software Extension Guard[2] (Intel SGX) technology available in the latest generation of Intel Xeon processors.

For Confidential Computing in code, you will need to write your TEE application leveraging the Intel SGX technology with the SDK for Intel SGX[3] or, preferably with the Open Enclave SDK[4], an open source project that drives towards a consistent API surface around enclaving abstraction, and supports portability across various TEE types, and thus flexibility in architecture.

You will also need to create a Kubernetes cluster on hardware that supports Intel SGX. Fortunately, Azure is the very first major cloud platform to support provisioning of SGX-enabled virtual machines (VMs) under the umbrella of Azure Confidential Computing[5] (ACC). You can create a Kubernetes cluster with one or multiple agent pool(s) running ACC VMs by instantiating DC-series VMs size run Ubuntu 16.04 or Ubuntu 18.04 on Intel Xeon processors and that are Confidential Computing ready.

So, you will then need to provision a Kubernetes cluster either by using Azure Kubernetes Services (AKS)[6] or AKS-Engine[7].

AKS is the Microsoft proprietary managed Kubernetes cluster toolbox, allowing for easily controlled and monitored cluster providing a true out-of-the-box experience. These clusters appear as a single resource on your

[1] Trusted execution environment: https://en.wikipedia.org/wiki/Trusted_execution_environment

[2] Intel Software Extension Guard: https://software.intel.com/en-us/sgx

[3] SDK for Intel SGX: https://software.intel.com/en-us/sgx/sdk

[4] Open Enclave SDK: https://github.com/openenclave/openenclave

[5] Azure Confidential Computing: http://aka.ms/azurecc

[6] Azure Kubernetes Service (AKS): https://azure.microsoft.com/en-us/services/kubernetes-service/

[7] AKS-engine: https://github.com/Azure/aks-engine

Azure portal, providing a total abstraction of all the inner workings of Kubernetes. The cluster can then be used as an incubator for your application without any additional configuration required.

AKS-Engine is (yet) another open-source project from Microsoft that provides convenient tooling to quickly bootstrap a Kubernetes cluster on Azure, and maintain clusters provisioned with basic IaaS resources in Azure by leveraging Azure Resource Manager[8] (ARM): AKS-Engine allows you generates Azure Resource Manager templates you can use for deploying Kubernetes clusters on Azure. With this project, you are in complete control of the created resources.

A Confidential Computing device plugin (manually installed if using AKS-engine, already installed if using AKS) will be then used to surface the usage of the Encrypted Page Cache (EPC) RAM as a schedulable resource for Kubernetes, and will allow you to schedule pods and containers that use the Open Enclave SDK onto hardware which supports TEE.

In this guide, and as its title suggest, we will cover the basics of the above, i.e. illustrating how SGX-capable nodes of a Kubernetes cluster and to deploy a TEE based application onto it.

For that purposes, you're invited to follow a short series of modules, each of them illustrating a specific aspect of the above outlined steps for leveraging Confidential Computing with Kubernetes on azure.

Each module within the guide builds on the previous. You're free to stop at any module you want, but our advice is to go through all the modules.

At the end of the starter guide, you will be able to:

- Create a Kubernetes cluster with AKS-Engine on top of DC-series VMs for SGQ-capable nodes, along with the Confidential Computing device plugin.
- Use the Open Enclave SDK to create a TEE based application that targets Intel SGX,
- Containerize the TEE based application,
- Deploy the TEE based application on the Kubernetes cluster with Confidential Computing.

Et voilà!

Guide elements

In the guide modules, you will see the following elements:

- **Step-by-step directions**. Instructions or links to online documentation for completing each procedure or part.
- **Important concepts**. A short explanation of some of the concepts important to the procedures in the module, and what happens behind the scenes.

[8] What is Azure Resource Manager?: https://docs.microsoft.com/en-us/azure/azure-resource-manager/management/overview

- **Sample application, and files**. A downloadable or cloneable version of the project containing the code that you will use in this guide, and other files you will need. **Please go to https://github.com/openenclave/ on GitHub to download or clone all necessary assets.**

Guide prerequisites

To successfully leverage the provided code in this starter guide, you will need:

- A Windows or Linux local machine
- Docker, Azure-CLI , AKS-engine, kubectl[9] installed
- A Microsoft account[10] along with an Azure subscription. If you don't have an Azure subscription, create a free account[11] before you begin. **You must be able to create a service principal[12] with the Contributor role.**
- A fast and reliable Internet connection

So, it's high time to get our hands dirty with the keyboard! :-)

[9] Kubectl on Kubernetes: https://kubernetes.io/docs/reference/kubectl/kubectl/

[10] Microsoft Account: https://account.microsoft.com/account?lang=en-us

[11] Create your Azure free account today: https://azure.microsoft.com/en-us/free/?WT.mc_id=A261C142F

[12] Application and service principal objects in Azure Active Directory: https://docs.microsoft.com/en-us/azure/active-directory/develop/app-objects-and-service-principals

Module 1: Setting up a Kubernetes cluster

Deploying an application using Intel SGX capabilities requires to have at least one machine, a Kubernetes node, to be equipped with an SGX-enabled processor underneath.

There are two ways to achieve setting up the cluster: either by using the AKS-Engine or the Azure Kubernetes Service (AKS) service.

Let's start with the first one.

Alternative 1: Using AKS-Engine

In this first alternative, you will use AKS-engine, the underlying core behind a classical Azure AKS deployment.

Using AKS-Engine doesn't require any form submission or prior registration as you will see for AKS but is a bit more manual. However, this feature (in Alpha) should be considered as in a proof of concept state, and thus IS NOT intended for production purposes. It has been introduced prior to the above preview

Using the engine directly allows you to specify which type of VM to require to build the Kubernetes cluster. For the sake of this guide, you will be using the capabilities of Azure Confidential Computing (ACC), and creating v1 DC-Series VMs, and more particularly Standard_DC2s[13] VMs with 2 vCPUs and 8 GiB of memory.

> **Important Note** V1 DC-Series VMs are in Preview and deploy an older version of the DC-Series VMs. They aren't going to be generally available and will remain in preview until deprecation.
>
> For the most up-to-date technology and Confidential Computing VMs, you will need to use the new Generation 2[14] DCsv2-Series[15] instead that correspond to an Azure Confidential Compute (Virtual Machine) V2 deployment. Besides the initially released v1 DC-Series VMs, DCsv2-Series VMs are backed by the latest generation of Intel XEON E-2288G Processor with the Intel SGX technology.

The AKS-engine quick start for SGX[16] presents a method to build an SGX-enabled cluster, and is the basis used for this guide. Follow all the instructions provided.

To set up the cluster, a resource group must be created. This resource group will contain all of the cluster components.

An Azure Active Directory (Azure AD) service principal is then created for this resource group. This service principal will act as a surrogate to create all the cluster building blocks while deploying and must be created with a **Contributor** role.

[13] Preview: DC-series: https://docs.microsoft.com/en-us/azure/virtual-machines/sizes-previous-gen#preview-dc-series

[14] Support for generation 2 VMs on Azure: https://docs.microsoft.com/en-us/azure/virtual-machines/linux/generation-2

[15] Preview: DCsv2-series: https://docs.microsoft.com/en-us/azure/virtual-machines/dcv2-series

[16] Using SGX with Kubernetes: https://github.com/Azure/aks-engine/blob/master/docs/topics/sgx.md

The cluster model to deploy is contained in the file *deployment.json*. An example deployment can be found in the Appendix.

```
$resGroupName = "<Ressource group name>"
$resGroupLocation = "<Ressource group/cluster location>"
$modelToDeploy = "deployment.json"
$subName = "<Your subscription name here>"
# Get Azure subscription from the list and set it as default
$subscription = `
    az account list --query "[?name=='$subName']" | ConvertFrom-Json
az account set --subscription="$($subscription.id)"

# Create a resource group to receive the cluster elements
$groupDef = `
    az group create --name "$resGroupName" --location "$resGroupLocation" | ConvertFrom-Json

# Create service principal for the created resource group
$servicePrincipal = `
    az ad sp create-for-rbac --role="Contributor" --
scopes="/subscriptions/$($subscription.id)/resourceGroups/$($groupDef.name)"

# Actually deploy the cluster using the template in deployment.json
aks-engine deploy --subscription-id $($subscription.id) `
    --dns-prefix $resGroupName `
    --resource-group $resGroupName `
    --location $resGroupLocation `
    --api-model $modelToDeploy `
    --client-id $servicePrincipal.appId `
    --client-secret $servicePrincipal.password `
    --set servicePrincipalProfile.clientId="$($servicePrincipal.appId)" `
    --set servicePrincipalProfile.secret="$($servicePrincipal.password)"
```

Let's now consider the second alternatives with AKS.

Alternative 2: Using AKS

As already outlined, building an SGX-enabled cluster using AKS is a preview feature, but is the recommended way, as it will be the go-to solution in the near future.

As such, the preview of "Confidential workloads on Azure Kubernetes Service (AKS)" service, i.e. the ACC Node Pools support on AKS, has been announced as a preview feature at the //Build2020 Conference.

Leveraging the preview is a three steps process:

1. Being added to added to the preview.
2. Registering for the preview.
3. Creating the cluster.

Being added to added to the preview

As of this writing, you can join this preview if you are interested by submitting a form at https://aka.ms/accakspreview (https://microsoft.qualtrics.com/jfe/form/SV_1NU9xTXfFKUQXFH) with your details and your Azure Subscription ID that you plan to use with this starter guide.

Registering for the preview

Once accepted, you will first need to register to the preview. This is a simple step, only requiring a few commands.

You may however need to increase your ACC quota by creating a support ticket[17].

The sample cluster built during this guide shouldn't require increasing the quota, but any production environment using more than 1 or 2 ACC cores will require this quota to be increased.

You can then register to the AKS-preview for confidential computing using this Azure CLI script.

```
# This scripts register the current user for the private preview version of AKS

# Wait for a register call to be completed. Throw on abnormal states
function waitUntilRegistered($name, $namespace){
    enum ContainerServiceState{
        LOADING
        OK
    }
    $values = @{
        [ContainerServiceState]::LOADING = "Registering";
        [ContainerServiceState]::OK = "Registered"
    }

    do {
        $state = `
        (az feature show `
         --name $name `
         --namespace $namespace `
         | ConvertFrom-Json).properties.state
        if ($state -ne $values.[ContainerServiceState]::LOADING`
            -and $state -ne $values.[ContainerServiceState]::OK) {
            throw [System.Exception]::new("Couldn't register to the service")
        }
        Start-Sleep -s 5
    }until($state -eq $values.[ContainerServiceState]::OK)
}
# Register to AKS private preview
az extension add --name aks-preview
az feature register --name "Gen2VMPreview" --namespace "Microsoft.ContainerService" --verbose
waitUntilRegistered "Gen2VMPreview" "Microsoft.ContainerService"
# Refresh service to propagate changes
az provider register --namespace 'Microsoft.ContainerService'
```

Creating the cluster

Once you have been registered for the preview, creating the cluster is as simple as creating any other AKS cluster, the only difference being using a custom header (`usegen2vm=true`) to use Generation 2 VMs[18]. This is required because all ACC nodes are second generation VMs.

First, a resource group must be created. This resource group will contain all of the cluster components.

[17] Standard quota: Increase limits by VM series: https://docs.microsoft.com/en-us/azure/azure-portal/supportability/per-vm-quota-requests

[18] Support for generation 2 VMs on Azure: https://docs.microsoft.com/en-us/azure/virtual-machines/linux/generation-2

An Azure Active Directory (Azure AD) service principal is then created for this resource group. This service principal will act as a surrogate to create all the cluster building blocks while deploying and must be created with a **Contributor** role.

Finally, using a single `aks create` command, you can build a whole confidential computing-ready Kubernetes cluster!

You can use the script below to deploy a cluster named **brandNewCluster** within the Resource Group **sconaksrg**.

```
$resGroupName = "sconaksrg"
# Must be a region with support for ACC nodes. Either uksouth, canadacentral or eastus
$resGroupLocation = "uksouth"
$clusterName = "brandNewCluster"
# Every compatible nodes are following the "Standard_DCXs_v2" pattern, X being the number of vCPU
$nodeSize = "Standard_DC1s_v2"
# Standard ACC quota is 8. 2 nodes with 1 vCPU is using 2*1 = 2 on the 8 allowed
$nodeCount = 2

# ------------------------------------------------------------------------

# Create the cluster resource group
$groupDef = `
    az group create --name "$resGroupName" --location "$resGroupLocation" | ConvertFrom-Json

# Create a service principal for the resource group, to create the cluster
$servicePrincipal = `
   az ad sp create-for-rbac --role="Contributor" --
scopes="/subscriptions/$($subscription.id)/resourceGroups/$($groupDef.name)"

# Create the cluster
az aks create `
    -l $resGroupLocation `
    -g $resGroupName `
    -n $clusterName `
    --service-principal $servicePrincipal.appId `
    --client-secret $servicePrincipal.password `
    --vm-set-type VirtualMachineScaleSets `
    --node-count $nodeCount `
    --kubernetes-version 1.15.10 `
    --network-plugin azure `
    --node-vm-size $nodeSize `
    --aks-custom-headers usegen2vm=true

# Cluster created ! Get the kubeconfig to use with kubectl
az aks get-credentials --resource-group $resGroupName --name $clusterName

# Catch exception, log and exit
trap {
    Write-Log $PSItem.ToString()
    exit -1
}
```

For the sake of this guide, you will be using here for the two nodes the capabilities of Azure Confidential Computing (ACC), and more particularly DCsv2-Series[19] VMs with 1 vCPUs and 4 GiB of memory.

[19] Preview: DCsv2-series: https://docs.microsoft.com/en-us/azure/virtual-machines/dcv2-series

As of this writing, the DCsv2-Series VMs are available in Canada Central, UK South & US East regions, see DCsv2-series[20]. UK South is target here. However, please deploy to Canada Central if you run into any issues.

Regardless of the alternative you've opted to, if everything went smoothly, you should now have a Kubernetes cluster in place with SGX-capable nodes.

Let's now consider how to containerize a TEE based application written with aforementioned Open Enclave SDK.

[20] DCsv2-series: https://docs.microsoft.com/en-us/azure/virtual-machines/dcv2-series

Module 2: Containerizing a TEE based application

As you may know, only containerized applications can be deployed onto Kubernetes. Thus, every TEE based application makes no exception to that, and must be containerized before being deployed on your newly created Kubernetes cluster.

To achieve that, you are now invited to follow the Open Enclave SDK quick start guide[21] as a basis. It is, however, important to remember that as containers share an underlying kernel with their host, **Data Center Attestation Primitives (DCAP) drivers don't have to be installed inside the containers**.

> **Important Note** DCAP provides SGX attestation support for SGX enclave. For more information about DCAP and attestation, please refer to the guide **Leveraging attestation for TEE based applications on Azure** in this series of guides.

As of this writing, ACC VMs are installed with Intel SGX DCAP driver version 1.2.6[22]. Microsoft work closely with Intel to keep the patches updated.

It is also important to keep in mind that containers should be as lightweight as possible. Thus, only the compiled binary and its dependencies should be shipped into the cluster. A great way to achieve this is by using Docker multi-stage builds[23], as shown below for the Hello World sample[24], coming with the Open Enclave SDK, and being used here for illustration purpose.

```
FROM ubuntu:16.04 as build
RUN apt update && apt -y install wget\
    # Add Microsoft repositories and keys
    && echo 'deb [arch=amd64] https://download.01.org/intel-sgx/sgx_repo/ubuntu xenial main' | tee
/etc/apt/sources.list.d/intel-sgx.list\
    && wget -qO - https://download.01.org/intel-sgx/sgx_repo/ubuntu/intel-sgx-deb.key | apt-key add -
\
    && echo "deb http://apt.llvm.org/xenial/ llvm-toolchain-xenial-7 main" | tee
/etc/apt/sources.list.d/llvm-toolchain-xenial-7.list\
    && wget -qO - https://apt.llvm.org/llvm-snapshot.gpg.key | apt-key add -\
    && echo "deb [arch=amd64] https://packages.microsoft.com/ubuntu/16.04/prod xenial main" | tee
/etc/apt/sources.list.d/msprod.list\
    && wget -qO - https://packages.microsoft.com/keys/microsoft.asc | apt-key add -\
    # Update repo and install SGX lib
    && apt install -y apt-transport-https ca-certificates\
    && apt update\
    && apt -y install clang-7 libssl-dev gdb libsgx-enclave-common libsgx-enclave-common-dev
libprotobuf9v5 libsgx-dcap-ql libsgx-dcap-ql-dev az-dcap-client open-enclave

# Build the HelloWorld app
WORKDIR /app
COPY . /app
```

[21] Install the Open Enclave SDK (Ubuntu 18.04): https://github.com/openenclave/openenclave/blob/master/docs/GettingStartedDocs/install_oe_sdk-Ubuntu_18.04.md

[22] DCAP Linux 1.2 Open Source: https://01.org/intel-softwareguard-extensions/downloads/intel-sgx-dcap-linux-1.2-release

[23] Docker multi-stage builds: https://docs.docker.com/develop/develop-images/multistage-build/

[24] Hello World sample: https://github.com/openenclave/openenclave/tree/master/samples/helloworld

```
WORKDIR /app/samples/helloworld
# Note : sourcing openenclaverc actually add all the build tools into the PATH
RUN /bin/bash -c "source /opt/openenclave/share/openenclave/openenclaverc && make build "
#Stage 2 : Production
FROM busybox:glibc
# Copying the compiled hello world app
COPY --from=build /app/samples/helloworld /
# Copy over the shared lib it depends on
COPY --from=build /lib/x86_64-linux-gnu/libdl.so.2 /lib/
COPY --from=build /usr/lib/x86_64-linux-gnu/libsgx_enclave_common.so.1 /lib/
COPY --from=build /usr/lib/x86_64-linux-gnu/libsgx_dcap_ql.so.1 /lib/
COPY --from=build /usr/lib/x86_64-linux-gnu/libsgx_urts.so /lib/
COPY --from=build /usr/lib/x86_64-linux-gnu/libssl.so /lib/
COPY --from=build /lib/x86_64-linux-gnu/libssl.so.1.0.0 /lib/
COPY --from=build /lib/x86_64-linux-gnu/libcrypto.so.1.0.0 /lib/
COPY --from=build /usr/lib/x86_64-linux-gnu/libstdc++.so.6 /lib/
COPY --from=build /lib/x86_64-linux-gnu/libgcc_s.so.1 /lib/
CMD ["host/helloworldhost", "./enclave/helloworldenc.signed"]
```

Using multi-stage builds allows the final image size to be about 17 MB. A lot of smaller than a single stage build, that would be about 1.4 GB, accounting for Ubuntu and all the build tools downloaded.

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
eternalintern/enclavehello	busybox	11d855691ec8	6 days ago	16.6MB
eternalintern/enclavehello	latest	0828411dd381	6 days ago	1.43GB

Figure 1: multi-stage building size comparison

Please note that in the above Dockerfile, all the shared libraries used by the Hello World sample are copied over to the production stage (busybox). This technique isn't applicable for a large application with multiple dependencies, as it would be too difficult to maintain. To attain the smallest image size possible for larger deployments, using static compilation and a binary packer[25] would be a better solution.

Please also note that both the official Hello World image[26] (504 MB) and the Hello World image used in this guide[27] (17 MB) are available on the Docker Hub.

Now that you are equipped with container images for your TEE based application, let's now see how to adequately deploy it on your previously created cluster.

[25] UPX binary packer: https://github.com/upx/upx

[26] Official Hello World image: https://hub.docker.com/r/oeciteam/sgx-test

[27] Minimal Hello world image: https://hub.docker.com/r/eternalintern/enclavehello

Module 3: Deploying a TEE based application

The final step is to deploy the containerized TEE based application on the Kubernetes cluster.

Alternative 1: Using AKS-Engine

With AKS-Engine, there are two ways to achieve it, depending on whether you use or not a device plugin.

Fortunately, the AKS-Engine guide[28] explains both ways:

1. Direct deployment.
2. Using a device plugin.

Let's consider them successively so that you can ultimately opt for the best one :-)

Direct deployment

Your TEE application needs direct access to the Intel SGX capabilities. To achieve it, a privileged container[29] is used in conjunction with a volume mounting of the SGX descriptor /dev/sgx[30].

```
apiVersion: v1
kind: Pod
metadata:
  name: enclavehello
spec:
  restartPolicy: OnFailure
  containers:
  - name: enclavehello
    image: eternalintern/enclavehello:busybox
    imagePullPolicy: Always
    volumeMounts:
    - name: dev-sgx
      mountPath: /dev/sgx
    securityContext:
      privileged: true
  volumes:
  - name: dev-sgx
    hostPath:
      path: /dev/sgx
      type: CharDevice
```

This deployment can be pushed directly onto the cluster with a typical declarative `kubectl` command, as follows:

```
$ kubectl apply -f.
```

[28] AKS-engine guide on Intel SGX: https://github.com/Azure/aks-engine/blob/master/docs/topics/sgx.md#optional-using-oe-sgx-device-plugin-alpha

[29] Docker privileged containers description: https://docs.docker.com/engine/reference/run/#runtime-privilege-and-linux-capabilities

[30] SGX details: https://github.com/jovanbulck/sgx-step/blob/master/README.md

However, this solution presents multiple flaws and **is NOT recommended**.

First, using a privileged container is a bad practice that should be avoided when possible, as it can lead to major security issues. But maybe even more importantly, this configuration doesn't support having both nodes with SGX and nodes without it.

Indeed, in this configuration, the Kubernetes Scheduler has no way to know that pods using SGX must be executed on SGX-capable nodes. This could result in random failures when the scheduler makes the wrong guess. Thus, this is not advised for any non-testing environment.

Note The Hello World sample should be a Kubernetes Job rather than a Pod.

Using a device plugin

A slightly more complicated solution is to rely on a device plugin. However, this solution does allow to use an "hybrid" cluster with both SGX and non-SGX nodes, as special instructions will be used to schedule pods needing SGX into SGX-enabled nodes.

First, the SGX-capable nodes need to be both labeled and tainted[31]. A label identifies a node with a special ability, such as an SGX-enabled processor. A **taint** repels any pods that doesn't **tolerate** it. Thus, any Kubernetes pods that doesn't support SGX won't be scheduled on an SGX tainted node (a physical machine).

Labeling and tainting a node can be achieved with the following commands:

```
$ kubectl label nodes <node-name> tee=sgx
$ kubectl taint nodes <node-name> openenclave.io/sgx_epc_MiB=true:NoSchedule
```

Next, a device plugin will expose the SGX Encrypted Page Cache (EPC)[32] as a schedulable resource for Kubernetes, thus removing the need to run every single application needing SGX as a privileged container. This plugin is applied to nodes having both the label `tee=sgx` and the taint `openenclave.io/sgx_epc_MiB`.

```
apiVersion: apps/v1
kind: DaemonSet
metadata:
  name: oe-sgx-device-plugin
  namespace: kube-system
  labels:
    app: oe-sgx-device-plugin
spec:
  selector:
    matchLabels:
      app: oe-sgx-device-plugin
  template:
    metadata:
      labels:
        app: oe-sgx-device-plugin
    spec:
      tolerations:
      - key: openenclave.io/sgx_epc_MiB # Toleration of sgx taint
        operator: Exists
```

[31] Kubernetes Labels and taints: https://kubernetes.io/docs/concepts/configuration/taint-and-toleration/

[32] SGX Encrypted Page Cache : https://eprint.iacr.org/2016/086.pdf#page=58

```
        effect: NoSchedule
      containers:
      - name: oe-sgx-device-plugin
        image: "mcr.microsoft.com/aks/acc/sgx-device-plugin:0.1"
        command: ["/usr/local/bin/oe-sgx-device-plugin"]
        imagePullPolicy: IfNotPresent
        volumeMounts:
        - name: device-plugin
          mountPath: /var/lib/kubelet/device-plugins
        - name: dev-sgx
          mountPath: /dev/sgx
        securityContext:
          privileged: true
      volumes:
      - name: device-plugin
        hostPath:
          path: /var/lib/kubelet/device-plugins
      - name: dev-sgx
        hostPath:
          path: /dev/sgx
      nodeSelector:
        tee: sgx # Selecting by label
```

Once the plugin is deployed using `kubectl`, the Encrypted Page Cache size on the SGX-enabled node is now visible on each node.

```
$ (kubectl get nodes <node> -o json | ConvertFrom-Json).status.allocatable
```

```
attachable-volumes-azure-disk : 2
cpu                           : 2
ephemeral-storage             : 27932155039
hugepages-1Gi                 : 0
hugepages-2Mi                 : 0
memory                        : 7400596Ki
openenclave.io/sgx_epc_MiB    : 40
pods                          : 30
```

Figure 2: Schedulable resources table for an SGX-enabled node

The final step is to deploy the containerized Hello World TEE based application. This can be achieved by applying this file:

```
apiVersion: v1
kind: Pod
metadata:
  name: enclavehello
spec:
  tolerations:
  - key: openenclave.io/sgx_epc_MiB
    operator: Exists
    effect: NoSchedule
  restartPolicy: OnFailure
  containers:
  - name: enclavehello
```

```
    image: eternalintern/enclavehello:busybox
    imagePullPolicy: Always
    resources:
        limits:
            openenclave.io/sgx_epc_MiB: 10
```

This pod *tolerates* the SGX taint, so it can be scheduled on the SGX-capable node. As the Encrypted Page Cache (EPC) is a schedulable resource, it is also possible to limit the max requirable size for this pod.

This is the preferred way to leverage Intel SGX on an Azure Kubernetes cluster.

Let's see how to proceed with AKS.

Alternative 2: Using AKS

Using SGX plugin/daemon set is also the path taken in AKS. The SGX plugin/daemon set is directly deployed from the get-go, see section § *Creating the cluster* above. This removes any additional step left in using SGX on a Kubernetes cluster.

You can verify that SGX plugin/daemon set is effectively installed during the provisioning process.

```
# Cluster created ! Get the kubeconfig to use with kubectl
$ az aks get-credentials --resource-group $resGroupName --name $clusterName
$ kubectl get pods –all-namespace
```

A pod named `sgx-device-plugin-xxx` should be listed. If the `sgx-device-plugin` pod is missing, you can manually install it:

- If AKS version >=1.17:

```
$ kubectl apply -f https://raw.githubusercontent.com/Azure/aksengine/master/docs/topics/sgx/device-plugin.yaml
```

- If AKS version is <1.17:

```
$ kubectl apply -f https://raw.githubusercontent.com/Azure/aksengine/master/docs/topics/sgx/device-plugin-before-k8s-1-17.yaml
```

This is only required if the SGX plugin/daemon set is not showing up.

At this stage, your are now ready to deploy the containerized TEE based application (see section § *Module 2: Containerizing a TEE based application* above) like you normally do with AKS application deployments.

Sample deployment file for this TEE based application on AKS will look like this:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: enclavehello-deployment
spec:
  selector:
    matchLabels:
      app: enclavehello
```

```
replicas: 1
template:
  metadata:
    labels:
      app: enclavehello
  spec:
    tolerations:
    - key: kubernetes.azure.com/sgx_epc_mem_in_MiB
      operator: Exists
      effect: NoSchedule
    restartPolicy: OnFailure
    containers:
    - name: enclavehello
      image: eternalintern/enclavehello:busybox
      command: ["/usr/local/bin/oe-sgx-device-plugin"]
      imagePullPolicy: IfNotPresent
      resources:
        limits:
          kubernetes.azure.com/sgx_epc_mem_in_MiB: 10
```

You can also follow some examples from here[33].

[33] OE Samples: https://github.com/openenclave/openenclave/tree/master/samples

As a conclusion

As of this writing, using Intel SGX capable hardware on Kubernetes is an experimental feature with AKS_engine, and more concise ways to achieve it should be on their ways as illustrated with the current public preview of AKS. It is, however, very easy to use, even in its pre-release state and could lead to major improvements in the confidential computing field.

For Confidential Computing, along with ARM Trust Zone, Intel SGX could become a milestone for the establishment of a global standard security practice for data critical domains.

Leveraging an idiomatic way of achieving Confidential Computing is a requirement for future cloud-native applications. Some Microsoft initiatives like the Open Application Model (OAM)[34] could even bring Confidential Computing to any Cloud with any orchestrator, bringing data security and privacy by design to the Cloud landscape.

If your curiosity is sharpened, to find additional information on OAM, and rudr[35], a Kubernetes Implementation of OAM, please do not hesitate to consult (yet) another series of whitepapers New perspectives for cloud-native applications with the Open Application Model (OAM), and the Distributed Application Runtime (Dapr). This series can be downloaded at http://aka.ms/CloudNativeAppsFuture.

This concludes this guide. We hope you enjoyed this (guided) tour!

[34] Open Application Model home page: https://oam.dev/

[35] Rudr: https://github.com/oam-dev/rudr

Appendix

```
{
"apiVersion":"vlabs",
"properties":{
          "orchestratorProfile":{
          "orchestratorType":"Kubernetes",
          "OrchestratorVersion":"1.14.7"

},
"masterProfile":{
          "count":1,
          "dnsPrefix":"",
          "vmSize":"Standard_D2_v3"

},
"agentPoolProfiles":[
{
           "name":"agentpool1",
           "count":1,
           "distro":"acc-16.04",
           "vmSize":"Standard_DC2s"

},
{
           "name":"agentpool2",
           "count":2,
           "distro":"aks-ubuntu-16.04",
           "vmSize":"Standard_D2_v2"

}

],
"linuxProfile":{
          "adminUsername":"azureuser",
          "ssh":{"publicKeys":[{"keyData":""}]}

},
          "servicePrincipalProfile":{
                    "clientId":"",
                    "secret":""

          }

}
}
```

Microsoft France
39 Quai du Président Roosevelt
92130 Issy-Les-Moulineaux