Chapitre 4
Création et gestion d'images Docker

1. Création manuelle d'une nouvelle image

1.1 Objectif

Dans le chapitre précédent, nous avons étudié le fonctionnement de base de Docker en utilisant des images préparées à l'avance pour nos exemples : l'image `hello-world` pour les premiers tests, l'image `ubuntu` pour les approches interactives par la ligne de commande, et enfin l'image `nginx` pour les tests impliquant un processus serveur. Ces images ont été récupérées en ligne sur le registre Docker Hub.

Bien que cette façon de fonctionner puisse suffire à des mises en œuvre extrêmement simples, un moment survient où il devient nécessaire de créer ses propres images pour évoluer vers plus de complexité. C'est ce type de manipulation que nous allons illustrer dans le présent chapitre.

1.2 Approche

L'approche naïve pour créer sa propre image est d'utiliser la commande `commit` entrevu dans le chapitre précédent pour persister l'état d'un conteneur sous forme d'une nouvelle image, après avoir ajouté à celui-ci les modifications nécessaires. Pour cela, nous pourrions lancer par exemple un conteneur sur une image `ubuntu` comme base et installer un produit en ligne de commande dans le conteneur.

Ensuite, le conteneur obtenu serait sauvegardé sous forme d'une image qu'il serait possible d'instancier ensuite autant de fois que souhaité. Bref, nous aurions créé une image avec un processus intégré selon nos besoins.

1.3 Difficultés

Dans les précédentes éditions du présent ouvrage, cette méthode était montrée en détail, de façon à faire apparaître toutes les difficultés liées :

- Taille de l'image, car les modifications, même d'effacement, s'empilent dans les couches et les effets d'une installation manuelle, avec des étapes de compilation et des productions de fichiers temporaires, peuvent être importants.
- Difficulté de changer le processus à démarrer lors du lancement d'un conteneur, car par défaut c'est celui de l'image de base qui se lance.
- Besoin d'une réelle expertise sur le paramétrage de l'application, de façon à l'installer de manière optimale et surtout suffisamment souple pour qu'un lancement de conteneur permette de faire varier certains des paramètres.
- Problématique de sécurité à gérer, en particulier lié au fait d'utiliser un utilisateur avec des droits élevés dans l'image.
- Complexité de gestion du cycle de vie si l'image de base est faite pour fonctionner en mode interactif et que nous souhaitons mettre en œuvre un serveur, fait pour fonctionner en mode démon, en arrière-plan.

1.4 Conclusion

La mise en œuvre d'une image propre par cette méthode est extrêmement complexe. De plus, dans les nouvelles versions des images de base, plus restreintes pour des raisons d'efficacité, certaines dépendances nécessaires au déploiement d'une application serveur doivent elles-mêmes être installées au préalable, y compris des fonctionnalités de niveau système qui nécessitent pour elles également une compétence développée.

Comme on l'aura compris, il est fortement recommandé de faire réaliser cette opération de construction d'une image par une personne experte de l'application ciblée, de façon à obtenir une image propre. Or, cette mise en œuvre propre existe certainement déjà, sous la forme d'une image Docker officielle. De plus, les personnes en charge de cette image nous offrent la "recette utilisée" sous la forme d'un fichier contenant toutes les commandes nécessaires à une installation parfaite. Pourquoi faire autrement ?

2. Utilisation d'un Dockerfile

2.1 Intérêt des fichiers Dockerfile

La "recette" dont nous venons de parler se présente sous la forme d'un fichier texte nommé `Dockerfile`, utilisant une grammaire particulière à Docker. Ce fichier contient toutes les opérations nécessaires à la préparation d'une image Docker. Ainsi, au lieu de construire une image par des opérations manuelles dans un conteneur suivie par une commande `commit` (voire plusieurs si l'on procède en étapes) comme nous l'avons évoqué ci-dessus, nous allons pouvoir compiler une image depuis une description textuelle de ces opérations.

Voici par exemple le contenu du fichier `Dockerfile` correspondant à l'image officielle MongoDB en ligne 3.6 (un système de gestion de base de données NoSQL) telle qu'elle peut être retrouvée sur le registre Docker Hub à la date d'écriture de ce livre (version 3.6.19 de MongoDB, disponible sur https://registry.hub.docker.com/_/mongo/) :

```
FROM ubuntu:xenial

# add our user and group first to make sure their IDs get assigned
consistently, regardless of whatever dependencies get added
RUN groupadd -r mongodb && useradd -r -g mongodb mongodb

RUN set -eux; \
    apt-get update; \
    apt-get install -y --no-install-recommends \
           ca-certificates \
           jq \
           numactl \
    ; \
    if ! command -v ps > /dev/null; then \
           apt-get install -y --no-install-recommends procps; \
    fi; \
    rm -rf /var/lib/apt/lists/*

# grab gosu for easy step-down from root
(https://github.com/tianon/gosu/releases)
ENV GOSU_VERSION 1.12
# grab "js-yaml" for parsing mongod's YAML config files
(https://github.com/nodeca/js-yaml/releases)
ENV JSYAML_VERSION 3.13.1

RUN set -ex; \
    \
    savedAptMark="$(apt-mark showmanual)"; \
```

```
    apt-get update; \
    apt-get install -y --no-install-recommends \
           wget \
    ; \
    if ! command -v gpg > /dev/null; then \
           apt-get install -y --no-install-recommends gnupg dirmngr; \
           savedAptMark="$savedAptMark gnupg dirmngr"; \
    elif gpg --version | grep -q '^gpg (GnuPG) 1\.'; then \
# "This package provides support for HKPS keyservers." (GnuPG 1.x only)
           apt-get install -y --no-install-recommends gnupg-curl; \
    fi; \
    rm -rf /var/lib/apt/lists/*; \
    \
    dpkgArch="$(dpkg --print-architecture | awk -F- '{ print $NF }')"; \
    wget -O /usr/local/bin/gosu
"https://github.com/tianon/gosu/releases/download/$GOSU_VERSION/gosu-
$dpkgArch"; \
    wget -O /usr/local/bin/gosu.asc
"https://github.com/tianon/gosu/releases/download/$GOSU_VERSION/gosu-
$dpkgArch.asc"; \
    export GNUPGHOME="$(mktemp -d)"; \
    gpg --batch --keyserver hkps://keys.openpgp.org --recv-keys
B42F6819007F00F88E364FD4036A9C25BF357DD4; \
    gpg --batch --verify /usr/local/bin/gosu.asc /usr/local/bin/gosu; \
    command -v gpgconf && gpgconf --kill all || :; \
    rm -r "$GNUPGHOME" /usr/local/bin/gosu.asc; \
    \
    wget -O /js-yaml.js "https://github.com/nodeca/js-yaml/raw/$
{JSYAML_VERSION}/dist/js-yaml.js"; \
# TODO some sort of download verification here
    \
    apt-mark auto '.*' > /dev/null; \
    apt-mark manual $savedAptMark > /dev/null; \
    apt-get purge -y --auto-remove -o
APT::AutoRemove::RecommendsImportant=false; \
    \
# smoke test
    chmod +x /usr/local/bin/gosu; \
    gosu --version; \
    gosu nobody true

RUN mkdir /docker-entrypoint-initdb.d

ENV GPG_KEYS 2930ADAE8CAF5059EE73BB4B58712A2291FA4AD5
RUN set -ex; \
    export GNUPGHOME="$(mktemp -d)"; \
    for key in $GPG_KEYS; do \
```

```
            gpg --batch --keyserver ha.pool.sks-keyservers.net --
recv-keys "$key"; \
    done; \
    gpg --batch --export $GPG_KEYS > /etc/apt/trusted.gpg.d/mongodb.gpg; \
    command -v gpgconf && gpgconf --kill all || :; \
    rm -r "$GNUPGHOME"; \
    apt-key list

# Allow build-time overrides (eg. to build image with MongoDB
Enterprise version)
# Options for MONGO_PACKAGE: mongodb-org OR mongodb-enterprise
# Options for MONGO_REPO: repo.mongodb.org OR repo.mongodb.com
# Example: docker build --build-arg MONGO_PACKAGE=mongodb-
enterprise --build-arg MONGO_REPO=repo.mongodb.com .
ARG MONGO_PACKAGE=mongodb-org
ARG MONGO_REPO=repo.mongodb.org
ENV MONGO_PACKAGE=${MONGO_PACKAGE} MONGO_REPO=${MONGO_REPO}

ENV MONGO_MAJOR 3.6
ENV MONGO_VERSION 3.6.19
# bashbrew-architectures:amd64 arm64v8
RUN echo "deb http://$MONGO_REPO/apt/ubuntu
xenial/${MONGO_PACKAGE%-unstable}/$MONGO_MAJOR multiverse" | tee
"/etc/apt/sources.list.d/${MONGO_PACKAGE%-unstable}.list"

RUN set -x \
# installing "mongodb-enterprise" pulls in "tzdata" which prompts for input
    && export DEBIAN_FRONTEND=noninteractive \
    && apt-get update \
    && apt-get install -y \
           ${MONGO_PACKAGE}=$MONGO_VERSION \
           ${MONGO_PACKAGE}-server=$MONGO_VERSION \
           ${MONGO_PACKAGE}-shell=$MONGO_VERSION \
           ${MONGO_PACKAGE}-mongos=$MONGO_VERSION \
           ${MONGO_PACKAGE}-tools=$MONGO_VERSION \
    && rm -rf /var/lib/apt/lists/* \
    && rm -rf /var/lib/mongodb \
    && mv /etc/mongod.conf /etc/mongod.conf.orig

RUN mkdir -p /data/db /data/configdb \
    && chown -R mongodb:mongodb /data/db /data/configdb
VOLUME /data/db /data/configdb

COPY docker-entrypoint.sh /usr/local/bin/
ENTRYPOINT ["docker-entrypoint.sh"]
```

```
EXPOSE 27017
CMD ["mongod"]
```

Les lignes passées en gras correspondent aux actions auxquelles un administrateur sans connaissance experte de MongoDB aurait certainement pensé, ce qui montre, par différence, la richesse additionnelle du fichier descriptif de l'image officielle. Les opérations les plus simples mises en œuvre dans ce fichier de génération d'image sont les suivantes :

– La mise en place d'un utilisateur et d'un groupe dédiés.
– L'installation des dépendances.
– Un commentaire de rappel comme quoi une vérification du téléchargement devrait être réalisée (par très rassurant de voir que la sécurité est laissée pour plus tard dans une image officielle d'une version récente de la base de données NoSQL la plus utilisée dans le monde…).
– L'installation proprement dite de l'application serveur.
– La définition des volumes pour les données ainsi que pour la configuration.
– L'utilisation d'un `ENTRYPOINT` dédié, qui permet de spécifier le script à lancer lors du démarrage du conteneur (nous reviendrons plus longuement sur cette commande par la suite).
– L'exposition du port sur lequel le processus reçoit les commandes (ce qui nécessite de connaître le fonctionnement de ce processus).
– La mise en œuvre d'un processus par défaut.

Encore une fois, il aurait été possible de trouver toutes ces manipulations par une recherche dans la documentation ou sur les forums. Mais la présence d'un `Dockerfile` produit par des experts de MongoDB, tout en nous laissant valider les commandes, est un énorme avantage de Docker.

Remarque

Un regard critique reste nécessaire sur les fichiers `Dockerfile`, sous peine de ne pas être plus en sécurité que dans le cas de la simple instanciation d'une image de provenance inconnue. En particulier, tout appel à des URL externes (par `curl`, typiquement) devrait être soigneusement vérifié. C'est le sens du `TODO` relevé dans le fichier, et qui est un manque de sécurité qui devrait être vérifié sur une mise en production, au minimum en vérifiant que le domaine associé est sûr et que le contenu sur lequel l'URL pointe ne comporte pas de problème de sécurité.

2.2 Utilisation d'un fichier Dockerfile

Bien que, dans le cas d'une image officielle, le fichier `Dockerfile` serve surtout au producteur pour réaliser l'image et montrer publiquement la façon dont celle-ci a été réalisée (le lien entre les deux étant assuré en particulier par le fait que c'est Docker Hub qui s'occupe de compiler les images), il est tout à fait possible d'utiliser soi-même ce fichier pour créer une image.

- Sur la machine hôte, créez un répertoire nommé par exemple `mongogb`.
- Placez-vous dans ce répertoire.
- Recopiez les deux fichiers que vous retrouverez sur https://github.com/docker-library/mongo/tree/master/3.6

Remarque

*Cette URL est celle contenant le fichier `Dockerfile`, qu'on peut retrouver directement depuis la page du registre Docker Hub concernant l'image officielle de MongoDB. Le plus simple pour recopier les fichiers est de lancer une commande `wget` sur les adresses qui sont obtenues en cliquant sur le bouton **Raw** dans la page d'affichage par GitHub des contenus des fichiers.*

- Si nécessaire, affectez les droits de lecture sur les deux fichiers et les droits d'exécution sur `docker-entrypoint.sh`.

Le second fichier (en plus de `Dockerfile`) est le fichier `docker-entrypoint.sh` auquel fait référence le `Dockerfile` (commande `ENTRYPOINT`, que nous allons expliquer plus bas). Il est nécessaire pour que le conteneur lance le serveur MongoDB sous l'utilisateur dédié au lieu de `root`. Il utilise pour cela l'utilitaire `gosu` qui avait été précédemment installé par une commande `RUN` du `Dockerfile`.

La partie du contenu de ce fichier correspondant à cette commande est la suivante :

```
#!/bin/bash
set -Eeuo pipefail

(...)

    exec gosu mongodb "$BASH_SOURCE" "$@"

(...)

exec "$@"
```

- Lancez la commande de compilation de l'image.

Chapitre 3
Déploiement d'applications avec Kubernetes

1. Contexte

1.1 Objectifs généraux

1.1.1 Exploiter la plateforme Kubernetes mise en place

Les précédents chapitres se sont attachés à montrer les différentes manières de monter un cluster Kubernetes, ainsi qu'à expliquer dans les détails les opérations de maintenance et d'exploitation de ce cluster une fois mis en place. Jusqu'à maintenant, donc, les opérations montrées relevaient plutôt de profils système ou opérationnels, dont le rôle est d'assurer la disponibilité de l'infrastructure.

Dans le présent chapitre, nous allons basculer dans un autre rôle, qui correspond à un profil d'exploitant du logiciel supporté par la plateforme, et donc plutôt consommateur de celle-ci. En résumé, après être rentré dans le détail de comment Kubernetes fonctionne, nous allons nous pencher sur ce que nous pouvons en faire. L'usage principal de ce type de plateforme étant de déployer des applications pouvant passer à l'échelle, nous utiliserons une application exemple que nous installerons sur un cluster Kubernetes et que nous manipulerons comme le ferait l'exploitant d'une structure similaire en production.

1.1.2 Remarque sur l'approche DevOps

Dans son expression-même, "DevOps" peut faire penser que le principe associé est qu'une même personne possède les deux rôles de développeur et d'opérationnel, ou à l'inverse qu'il convient de bien séparer ces deux rôles. En fait, le principe du DevOps est de faire en sorte d'éviter les barrières entre les personnes en charge de l'infrastructure et celles en charge des logiciels portés par cette dernière.

Traditionnellement, les approches sont en effet opposées car les enjeux sont très différents : pour l'opérationnel en charge de la plateforme, l'enjeu principal est que celle-ci fonctionne en continu, ce qui a pour conséquence naturelle une certaine aversion au changement. Après tout, un des commandements principaux dans l'informatique est de ne pas toucher à un système qui fonctionne. Tout changement est potentiellement vecteur d'un nouveau bug, d'un problème dans la procédure de mise à jour et n'importe quel administrateur système a donc comme réflexe de réduire au maximum ces interventions.

À l'inverse, les développeurs et les profils qui commanditent ces changements, que ce soient des Product Owners sur un produit ou le client/utilisateur en direct sur un projet, ont plutôt intérêt à ce que les fonctionnalités sortent le plus vite possible, pour apporter le maximum de valeur métier au logiciel. À l'inverse de l'administrateur du système, ils souhaitent donc généralement accélérer le rythme du changement. Cette différence dans les enjeux a longtemps été traitée par un rapport de force et une opposition entre les deux approches, les seconds obtenant bon gré mal gré des premiers des mises en production tandis que ceux-ci invoquaient toutes les raisons, bonnes ou mauvaises, pour ne pas les réaliser.

Comme le périmètre de recoupement entre les deux responsabilités est traditionnellement assez large, les difficultés étaient plus étendues. Par exemple, un développeur pouvait avoir besoin d'un OS particulier pour son application, alors que l'administrateur souhaitait ne plus le supporter. Dans certains cas, un administrateur pouvait décider de blocages de sécurité trop contraignants pour un applicatif. Bref, le champ d'interaction était très large entre les deux rôles, car il couvrait toutes les technologies de support de déploiement, de la machine physique jusqu'au choix de librairies partagées.

Une approche plus évoluée est de s'attaquer au problème lié à l'incertitude sur le changement elle-même. Les approches Lean et Agile recommandent de réaliser les changements les plus petits et granulaires possibles, de façon à ce que l'impact soit le plus limité possible. Une bonne approche d'assurance qualité et de tests automatisés renforce cette sécurité sur le changement. Enfin, les principes d'intégration et de déploiement continu se sont attaqués à un autre pan du problème, en faisant en sorte qu'une mise en production devienne une tâche la plus banale possible.

Pour arriver à ce résultat, et lutter contre l'impression de "balancer une release par-dessus le mur", l'automatisation est certes une part de l'équation, avec la capacité de revenir facilement en arrière, mais il est également important de travailler sur les aspects management et méthodologie. C'est l'approche DevOps qui trouve son origine dans cette volonté de rendre les interactions simples entre les développeurs et les administrateurs de la plateforme logicielle déployée. Or, en plus d'un état d'esprit à insuffler, il convient de disposer également d'outils rendant possible et simple ce partage d'une réalité qui doit être gérée en commun au lieu de faire l'objet de conflits.

Kubernetes s'inscrit fortement dans cet objectif véhiculé de manière plus globale par les approches de gestion par conteneurs en fournissant une solution qui permet aux deux rôles de se répartir au mieux les tâches tout en leur offrant une réalité partagée, un contrat simple qui leur permet de travailler ensemble, mais en offrant le moins possible de prise à des conflits.

Ce qu'attendent principalement ces rôles d'une plateforme Kubernetes est qu'elle les rendent les plus indépendants possible des technologies liées à l'infrastructure : gestion du réseau physique, des machines physiques ou virtuelles, de l'ajout de ressources matérielles avec tout ce que cela comporte en termes de branchement, mais aussi de ressources "secondaires", comme la climatisation ou l'électricité, qu'il faut en outre sécuriser sur leur approvisionnement. L'idée est donc que l'utilisateur du cluster Kubernetes puisse simplement connaître l'adresse de celui-ci et les informations nécessaires pour s'y connecter, mais qu'il soit le moins possible exposé à cette complexité qui est liée au rôle de l'administrateur.

Le contrat résultant entre le producteur de la plateforme et le client de celle-ci repose au final uniquement sur l'adresse du cluster Kubernetes (nous reviendrons plus loin sur les portions techniques de cette "adresse" ou "identifiant", en reprenant les concepts de contexte, cluster et espace de nommage dans le cadre de l'exploitation logicielle de la plateforme Kubernetes). Le rôle d'administrateur consiste à s'occuper de la disponibilité des ressources matérielles, de la robustesse du cluster, de son élasticité, etc. et fournit une adresse pour le cluster. Le rôle d'exploitant logiciel utilise cette adresse pour déployer des applicatifs dans une configuration telle que souhaitée, et ce sans dépendance à l'administrateur autre que la connaissance de l'adresse du cluster. Il s'occupe des technologies à mettre en place dans le conteneur Docker, des librairies en dépendance, de la sécurité logicielle, etc., et fournit au cluster une boîte noire qui expose un port réseau avec des fonctionnalités en échange de ressources livrées de manière standard.

Dans la pratique, il est évident que des échanges sont nécessaires en plus, ne serait-ce que pour équilibrer les coûts du cluster avec les gains attendus par l'application que celui-ci supporte. Toutefois, ces enjeux sont désormais suffisamment éloignés de la technique pour qu'ils ne constituent pas un sujet de conflit immédiat. L'ajout de ressources étant simple – et surtout sans aucun impact sur la partie logicielle, il est relativement aisé de fournir une plateforme à coût réduit et de la faire grossir au fur et à mesure des besoins justifiés économiquement par les gains du logiciel. Cette souplesse rend également moins conflictuel le rapport entre les deux parties.

1.1.3 Principaux enseignements à attendre

Comme le présent chapitre est axé sur le consommateur de la plateforme, les sujets principaux relèveront de la façon de structurer une application pour la déployer sur Kubernetes, puis sur les méthodes utilisables pour réaliser ce déploiement et enfin sur celles à disposition pour maintenir le logiciel dans un état fonctionnel.

De façon à rendre plus compréhensibles tous les points d'attention (et ils sont nombreux) pour réaliser ceci, une première manipulation sera réalisée de manière manuelle et la plus détaillée possible, pour montrer tous les recoins de la technologie et bien expliquer les concepts associés.

Ensuite seulement, un second exemple montrera une réalisation plus automatisée et donc plus conforme à ce qu'on peut s'attendre à voir dans la réalité, sur des systèmes en production.

Kubernetes est souvent présenté comme une plateforme qui permet de simplifier fortement le déploiement, les mises à jour et les passages à l'échelle des applications web, mais cette simplicité ne va pas sans un coût initial pour la masquer. Ce coût est en partie supporté par la technologie elle-même, mais également par le paramétrage de l'application cible en amont. Bref, Kubernetes rend effectivement plus simple et robuste la vie d'une application, mais cela suppose de traiter une partie de la complexité inhérente à cette activité lors du montage de la solution. C'est toute cette complexité que nous allons détailler ci-dessous, de façon qu'une fois la plateforme fonctionnelle, elle le soit de manière extrêmement robuste et évolutive.

1.2 Outillage

1.2.1 Cluster Azure Kubernetes Services

Il ressort de toute l'introduction au présent chapitre qu'un cluster Kubernetes doit être prêt pour une utilisation de façon à réaliser les exercices ci-dessous. Nous pourrions laisser à l'utilisateur le choix du mode d'installation vu que plusieurs ont été montrés auparavant. Toutefois, deux raisons font que nous reviendrons sur une installation d'un cluster.

La première est que la première partie du présent ouvrage est entrée en détail dans la structure d'un cluster Kubernetes, en expliquant les détails de mise en œuvre et toutes les étapes nécessaires aux réalisations les plus complexes. Toutefois, il existe également des méthodes pour bénéficier d'un cluster "clé en main", et ce suite à quelques clics sur des interfaces web simples. Comme il était nécessaire de disposer d'une plateforme pour les exemples, autant en profiter pour montrer une nouvelle approche, qui constituera une corde de plus à l'arc de l'administrateur, bien que très rudimentaire. Il aurait bien sûr été possible d'utiliser Google Kubernetes Engine, mais comme Kubernetes vient lui-même de Google, la séparation entre ce qui vient de la plateforme et ce qui est fourni par le cloud de Google aurait été plus complexe à tracer, comme expliqué en introduction.

La seconde raison est que certains lecteurs ne seront potentiellement pas intéressés par l'installation ou le maintien en condition opérationnelle d'un cluster Kubernetes. Il ne serait donc pas approprié de les renvoyer à ces sections. Une solution rapide et efficace devait donc être fournie pour que ce type de lecteur puisse obtenir rapidement une plateforme pour ses tests, et se concentrer ensuite comme il le souhaite sur l'utilisation de Kubernetes elle-même.

Le choix des auteurs s'est porté sur la plateforme Azure et en particulier le service Azure Kubernetes Service, abrégé à partir de maintenant en AKS, pour sa simplicité de mise en œuvre. Microsoft fournit en effet plusieurs offres gratuites sur Azure. Celles-ci sont bien sûr limitées en ressources, mais elles ont l'avantage supplémentaire pour certaines d'entre elles de ne même pas nécessiter de carte de crédit pour s'inscrire, ce qui facilite l'inscription et réduit à zéro un éventuel risque de coûts cachés en cas de dépassement. La très bonne intégration avec la forge logicielle Azure DevOps et le registre de conteneurs Azure sont également des arguments pour commencer sur ce fournisseur de cloud.

Remarque

Par souci de transparence, le fait qu'un des auteurs soit Microsoft Most Valuable Professionnal sur Azure et bénéficie de nombreuses ressources gratuites pour les tests liés à l'écriture de ce livre a également pesé dans le choix du cloud de Microsoft. Des offres gratuites existent également sur le cloud d'Amazon ou sur le Google Container Engine/Google Kubernetes Engine, mais leur limitation en taille rendait difficile la mise au point de tous les exercices du présent ouvrage.

L'objectif, ici, étant de montrer la solution la plus simple et rapide possible pour bénéficier d'un cluster Kubernetes fonctionnel, nous passerons par l'interface graphique pour piloter la création de la ressource dans Azure. Cette interface est disponible sur https://portal.azure.com.