

PoC #0 - Using a Jenkinsfile configuration (pipeline)

Steps:

**1) Create and Configure a Jenkins Instance on Azure**

* Azure portal → Resouce groups → Create resource "CIAMProof"
* Azure portal → App services → create app service → Jenkins (Microsoft)
* Azure portal → Container registries → Add new  called CIAMProofACR linked to resource group "CIAMProof"
* Azure portal → Kubernetes services → Add new called CIAMProofAKS linked to resource group "CIAMProof"
* To access the Jenkins instance  - on windows terminal, run ssh -L 127.0.0.1:8080:localhost:8080 [ciam@ciamgigya.westeurope.cloudapp.azure.com](mailto:ciam@ciamgigya.westeurope.cloudapp.azure.com)
  + Password: CIAMTeam#2018
* This will create a tunnel linked to your local computer. On browser: [http://localhost:8080](http://localhost:8080/)
* Follow instructions (install suggested plugins)
* Once is installed, There are a number of plugin dependencies required. To check plugins in the Jenkins interface choose 'Manage Jenkins' from the menu on the left hand side and then select 'Manage Plugins'  
  Check that the following plugins are installed (if not, install them manually- plugins can be manually installed through the Jenkins UI)
  + Azure AD Plugin
  + Azure App Service Plugin
  + Azure CLI Plugin
  + Azure Commons Plugin
  + Azure Container Agents Plugin
  + Azure Container Service Plugin
  + Azure Credentials
  + Azure VM Agents
  + Bitbucket Branch Source Plugin
  + Bitbucket Build Status Notifier Plugin
  + Bitbucket Pipeline for Blue Ocean
  + Bitbucket Plugin
  + Bitbucket Pullrequest Builder Plugin
  + Blue Ocean (The UI for Jenkins)
  + BlueOcean Aggregator
  + Blue Ocean Core JS
  + Blue Ocean Pipeline Editor
  + Green Balls (Changes the positive colour of buttons from blue to green)
  + Pipeline
* Go to Jenkins → credentials to create a new credential of type "SSH Username with private key"
  + Username: [iban.balasch@es.nestle.com](mailto:iban.balasch@es.nestle.com)
  + Private key (enter directly):

-----BEGIN RSA PRIVATE KEY-----  
MIIEpAIBAAKCAQEAxDMX7X+c9h/uL5PZx8zvHrcV81lzG4VimlKtv8kS0ar/YHHl  
XxKKNhTEgkHJ8zbxdAmq1A0px7bi03dRir65297xHOlkXtBGIVzZi2xwcmOyaG67  
CgTHYeuGgPdOSavRz0B5P4H87nwl5L3TVrdJVkPE7j2+lrPXdYsXbVuT05sttq45  
C4Rpo1TN7Iky4tIitdJW3O9HtUDTlF4w6kfqCcceFNN27CTa0g1Ge+Gm/wm1y3Op  
iBcO9fdSD0yWhGDaLVUfVXiN23DkmXrjwF5hJctroqR0a/2PVNLGJZ6nzZqs2J8u  
Ix5xL3xt+pNYNyY+AgQRLFxO3Quq16s+9Az2IwIDAQABAoIBAQCSe5snxJlJ4TkR  
6I82bzkwc309/NDGAWZG1efHQrZ8N/EG7sJToyHxkrvMjJXcsm6/NeCHBXTmGRNs  
0q7cqd1V3TOjZFV9vH4/tnTtdX7WyLyQ3+thZYP/BbucWgfnVRMGa5qa4dAXtjUM  
fPVmzt1APMSxIwRCu+7nYXuSOEc8OlZU1P2Bpcv2E/r9AxHw3Z3n5dsB10UnoqPX  
EXS5kCnJ/kPT9BTGiu/xj1f7cDadJEtHiXdVGtmcWnZPNR0ePTHo+8sCMaCrN5EH  
/YeYEdO/ql2RNeRnOTFF0epEX/q3jGY7EUWbeW6QBZ5XHPtTINS/qvOnq9NAsbjd  
b8bRQhrxAoGBAOzCtgWUuazCLCpPX/PDDYiF/DyFmhXvgf6QgJEPRJTXooHgHdRJ  
ne1yR9A9ZY2v7MeLboNrcWUXJ/DP4NiqkMqSqI32biBGMeLkeqcftPRKXAwzj9sP  
Bil/cNI2WI5xZjmk5Cw9dro/nPiLy2HbmwafaGfFQIqLAdcc3JgFEA33AoGBANQk  
mVVSbEgvu5FpdWM5mGNebvc3H5UfaXL9lKEQYobLSEWttFLQEUy4BHJY78GimZXR  
/5vhY5HUEt5l5lXVEikqqMerLEY79Mjj5baDdcQI6SKT8p0/+RRzREF7N5RzKCqg  
hsyhvSJSGMnnecCtVD1QzM3LBVEhfbpT5CIZWf41AoGASk/LBfFqXdkMN7N2MHJ2  
WvWC+BQPCveD27zLPbyHEBimfxEzC6CXmyGVfLzolRI0xWT0KPeePdaIyMLWlerj  
S1HA6d4BLswZeFeyT6ZaC5sTTRkZ++25R2kipjg8j2j0Zv9kRqZ8D3VL9Y8bJPni  
rK93+6oj2dhLmZMLggOEyH0CgYA8Uxv3EzUPYeNoeiCTUrFusE/FpLsqYIBP3qQB  
GPOreOrYRYjdc6U3t6wsB/ZFjcM2wzx4SyNROIkVDMzu42QCqMrN3HL+ldYJuiR8  
Pe2jDRWea4u2A5cEpbJnHG61m/l0BanPTr7Fn6rE3znT/HXT6cITUwP0Ch4CehCJ  
UcQxcQKBgQDVqZjhaJHKLaOVSjFSdAGxB3d06aghDEb032j3s5ajiXrUi/0OH2C8  
Dt2LNNWCy+U7xxBTr/OynSWbZKrYr0kmvLoDAes+p+5HtKj6Es28bkp7kJvLyRk2  
C4HsQBW+xZRZF6ERWPJ+gdFOmZuSiFLWxAUWdcIP/g8y4Ym5N58RFw==  
-----END RSA PRIVATE KEY-----

* + ID: BitBucketAccessKey
  + Description: DSU Bitbucket access key (read and write)
* Create a new task at Jenkins ("Freestyle project"):
  + Select "Git" in "Configurar el origen del código fuente" section with these information:
    - Repository URL: <ssh://git@52.166.71.39:7999/ciam/schemavalidator.git>
    - Credentials: Select the "BitBucketAccessKey" create above
    - Specify branch (\*/master by default)
  + In "Disparadores de ejecuciones" select "Build when a change is pushed to BitBucket" option
  + In "Ejecutar" add a new step called "Deploy to Azure Container Service (AKS)":
    - Select credentials "azure\_service\_principal (local MSI)"
    - Select the resource group and container associated to this account
    - In config files write "Jenkinsfile"
    - Press "Verify configuration" to test that all works fine
  + Save task
* Return to the windows terminal (that is in "bash" mode connected into the Jenkins VM -via ssh-) and follow this steps to install the requirec CLIs:
  + Login into Azure AD and follow the instructions:
    - az login
  + Set the default subscription (in this POC, the subscription is CIAM - Dev Core Tools" (subscription id: b0122ca4-b7a9-4e33-9cdb-fa965a7849a0) so the command is:
    - az account set --subscription b0122ca4-b7a9-4e33-9cdb-fa965a7849a0
  + Install docker into Jenkins VM, the command is:
    - sudo apt install [docker.io](http://docker.io/)
  + Install Kubectl CLI
    - sudo snap install kubectl --classic
  + Install NPM:
    - sudo apt install npm
  + Restart Jenkins service:
    - sudo service jenkins restart
* Give permissions to Docker demon:
  + sudo usermod -a -G docker jenkins
  + sudo chmod 664 /var/run/docker.sock

**2) Azure Container and Azure Kubernetes**

* Return to the windows terminal (bash) again to follow these steps:
  + Enable admin into ACR:
    - az acr update -n CIAMProofACR --admin-enable true
    - The response will be like:

**Result**

|  |
| --- |
| {    "adminUserEnabled": true,    "creationDate": "2018-06-29T09:55:10.019926+00:00",    "id": "/subscriptions/b0122ca4-b7a9-4e33-9cdb-fa965a7849a0/resourceGroups/CIAMJenkinsResource/providers/Microsoft.ContainerRegistry/registries/CIAMP  roofACR",    "location": "westeurope",    "loginServer": "ciamproofacr.azurecr.io",    "name": "CIAMProofACR",    "provisioningState": "Succeeded",    "resourceGroup": "CIAMJenkinsResource",    "sku": {      "name": "Basic",      "tier": "Basic"    },    "status": null,    "storageAccount": null,    "tags": {},    "type": "Microsoft.ContainerRegistry/registries"  } |

* + Link AKS with ACR
    - Get assignee:
      * az aks show --resource-group CIAMProof --name CIAMProofAKS --query "servicePrincipalProfile.clientId" --output table
      * In this test, the value is: cb5a0bbd-905c-4bb4-b4b0-c630f20d5a5f
    - Get subscription:
      * az acr show --name CIAMProofACR --resource-group CIAMProof --query "id" --output table
      * In this test, the value is: /subscriptions/b0122ca4-b7a9-4e33-9cdb-fa965a7849a0/resourceGroups/CIAMProof/providers/Microsoft.ContainerRegistry/registries/CIAMProofACR
    - Set, to AKS, the role of Reader into ACS:
      * az role assignment create --assignee cb5a0bbd-905c-4bb4-b4b0-c630f20d5a5f --role Reader --scope /subscriptions/b0122ca4-b7a9-4e33-9cdb-fa965a7849a0/resourceGroups/CIAMProof/providers/Microsoft.ContainerRegistry/registries/CIAMProofACR
  + ACR secrets:
    - Get credentials from ACR:
      * az acr credential show -g CIAMProof --name CIAMProofACR
      * In this test, the value of the id "password" is "P2orn0wCYpWtY=ZmKEeXANzIFk68of3X" and "username" is "CIAMProofACR"
    - Create secret
      * kubectl create secret docker-registry ciamproofacr --docker-server [ciamproofacr.azurecr.io](http://ciamproofacr.azurecr.io/) --docker-email [iban.balasch@es.nestle.com](mailto:iban.balasch@es.nestle.com) --docker-username=ciamproofacr --docker-password "P2orn0wCYpWtY=ZmKEeXANzIFk68of3X"
      * In this test, the secret is "secret/ciamproofacr"
  + Create different namespaces for each environment, in AKS:
    - kubectl create namespace <environment>
* Push schemavalidator image into ACR:
  + We have a local docker image called schemavalidator:development, lets create a tagged version pointing to our ACR:
    - docker tag schemavalidator:development [ciamproofacr.azurecr.io/schemavalidator:development](http://ciamproofacr.azurecr.io/schemavalidator:development)
  + Ensure that we are logged into ACR:
    - az acr login --name CIAMProofACR
  + Push image into ACR:
    - docker push [ciamproofacr.azurecr.io/schemavalidator:development](http://ciamproofacr.azurecr.io/schemavalidator:development)
* Return to Jenkins, and modify the task and fill the required information for the section "Deploy to Azure Container Service (AKS)":
  + In the "Docker container registry credentials" subsection:
    - Kubernetes Namespace for Secret: secret
    - Secret Name: ciamproofacr
    - Credentials:
      * Docker registry url: [ciamproofacr.azurecr.io](http://ciamproofacr.azurecr.io/)
      * Credentials:  ...
* Jenkinsfile is a config file required by Jenkins to deploy it. We will configure it as pipeline.
  + Ensure that we are in the "default" namespace into Kubernetes (AKS):
    - kubectl config set-context $(kubectl config current-context) --namespace=default
* Copy .kube config file into jenkins folder:
  + First, we will re-create the credentials file (stored in the config file):
    - az aks get-credentials --name CIAMProofAKS --resource-group CIAMProof -a
  + This file is stored in the home folder, we need to move it to the jenkins folder (/var/lib/jenkins/.kube):
    - cd /var/lib/jenkins/.kube
    - sudo cp /home/ciam/.kube/config .
  + We will need to edit the config file because, on our tests, the context "CIAMProofAKS" by name, was duplicated so removed the duplicated context. The contexts remains like:

| **contexts:  - context:  cluster: CIAMProofAKS  namespace: default  user: clusterUser\_CIAMProof\_CIAMProofAKS  name: CIAMProofAKS  - context:  cluster: CIAMProofAKS  user: clusterAdmin\_CIAMProof\_CIAMProofAKS  name: CIAMProofAKS-admin  current-context: CIAMProofAKS-admin** |
| --- |

* sudo kubectl get services --watch -n development

PoC #1 - Deploy a Nexus service

Before create our CI/CD environment we created a PoC to test the best implementation using Azure as PaaS. On this case, we will try to install a Nexus instance into Kubernetes.

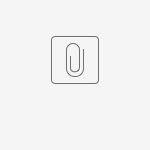
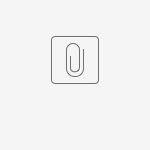
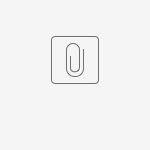
To realize this PoC we used the Microsoft's documentation about Kubernetes located at  <https://docs.microsoft.com/en-gb/azure/aks/tutorial-kubernetes-prepare-app>, the points 6 to 19 was done following this tutorial.

Let's prepare our local environment to make this proof. We will need to get a Nexus docker image from Sonatype's Github repository and deploy it in a local folder:

1. Install Docker for Windows application in your local machine (this is a Windows OS PoC)
   1. <https://www.docker.com/docker-windows>
2. Open a Powershell terminal window, move to the Nexus's folder and run the docker image:
   1. docker run -d -p 8081:8081 --name nexus sonatype/nexus3 (if nexus3 docker image doesn't exist locally, will be downloaded from Docker Hub main repository)
   2. Another approach to create a Docker image is using persistant data (recommended) but not was tested on this PoC. To test it, instead execute the command above, make this other steps:
      1. docker volume create --name nexus-data
      2. docker run -d -p 8081:8081 --name nexus -v [nexus-data:/nexus-data](http://nexus-data/nexus-data) sonatype/nexus3
3. In a browser's windows, navigate to [http://localhost:8081](http://localhost:8081/) to check is working correctly
4. Check that "nexus" appears as an image in Docker:
   1. docker images
5. Create a new resource group into our Azure's subscription:
   1. az group create --name CIAMContainerTest --location westeurope
6. Create a new Azure Container Service (ACS) to store our Docker's images:
   1. az acr create --resource-group CIAMContainerTest --name CIAMContainerACR --sku Basic
7. Login with your AAD account to be validated versus the ACS:
   1. az acr login --name CIAMContainerACR
   2. This command will give you an URL and a validation code to perform the login process
8. Get the loginServer attribute:
   1. az acr list --resource-group CIAMContainerTest--query "[].{acrLoginServer:loginServer}" --output table
9. Tag your local docker image with the loginServer attribute as prefix and specify the docker's inage current version (on this case, v1 is ok):
   1. docker tag nexus <loginServer>/nexus:v1
   2. In our PoC, the value of that attribute is "[ciamcontaineracr.azurecr.io](http://ciamcontaineracr.azurecr.io/)" so the command was: docker tag nexus [ciamcontaineracr.azurecr.io/nexus:v1](http://ciamcontaineracr.azurecr.io/nexus:v1)
10. Upload our Docker's image to ACS:
    1. docker push <loginServer>/nexus:v1
    2. In our PoC, the value of that attribute is "[ciamcontaineracr.azurecr.io](http://ciamcontaineracr.azurecr.io/)" so the command was: docker push [ciamcontaineracr.azurecr.io/nexus:v1](http://ciamcontaineracr.azurecr.io/nexus:v1)
11. Check that the image was uploaded and recognized by ACS
    1. az acr repository list --name CIAMContainerACR --output table
    2. You can check too if the tag was recognized with  az acr repository show-tags --name CIAMContainerACR --repository nexus --output table
12. Meanwhile Azure Kubernetes Service (AKS) is in preliminar mode, we need to registry it in our subscription. This must be executed once time per subscription
    1. az provider register -n Microsoft.ContainerService
13. Install the Azure Kubernetes CLI for Windows required to perform actions with that service:
    1. az aks install-cli
14. Create a new cluster service called CIAMClusterAKS:
    1. az aks create --resource-group CIAMContainerTest --name CIAMClusterAKS--node-count 1 --generate-ssh-keys
    2. We need to specify the number of nodes when we create our cluster BUT can be scaled (adding more CPUs) to our resource group
    3. **Important**: Each POD that we will create in Kubernetes will require a node
15. Login with AAD account to be validated versus the AKS:
    1. az aks get-credentials --resource-group CIAMContainerTest --name CIAMClusterAKS
16. Get the client ID for AKS (used in next steps)
    1. az aks show --resource-group CIAMContainerTest --name CIAMClusterAKS--query "servicePrincipalProfile.clientId" --output tsv
17. Get the resource ID for ACS:
    1. az acr show --name CIAMContainerACR --resource-group CIAMContainerTest--query "id" --output tsv
18. Create the roles between our AKS cluster and our ACS container repository:
    1. az role assignment create --assignee <id\_retrieve\_in\_step\_17>--role Reader --scope <path\_retrieved\_in\_step\_18>
19. Enable admin access in our ACS:
    1. az acr update -n CIAMContainerACR --admin-enable true
20. Get the credential (aka password) from our ACS:
    1. az acr credential show -g CIAMContainerTest --name CIAMContainerACR
    2. On this case, our credential value is "/G6zJp3gUaTXQNePretKehVfdB8F7TRu"
21. Create POD credentials in Kubernetes
    1. kubectl create secret docker-registry ciamcontaineracr --docker-server [ciamcontaineracr.azurecr.io](http://ciamcontaineracr.azurecr.io/) --docker-email [iban.balasch@es.nestle.com](mailto:iban.balasch@es.nestle.com) --docker-username=ciamcontaineracr --docker-password "/G6zJp3gUaTXQNePretKehVfdB8F7TRu"
    2. **Important**: the container name must be in LOWER case, otherwise will not work
22. Create the YAML file for our Nexus Docker image (located in our ACS container repository) to be deploy it on our AKS cluster.
    1. Due this is something to be explained  with deep details, please see the section "Yaml" on this document.
    2. Let's continue our PoC's explanation assuming that we already have that file correctly defined.
23. Create a Kubernetes's POD using our Nexus Docker image (from ACS):
    1. kubectl create -f nexus.yaml
24. Wait until POD is created and the service is exposed and assigned a public IP.
    1. To know the progression, you can run the command kubectl get service --watch
25. In a browser window, navigate to the public IP assigned to our Nexus repository to test that all works.

PoC #2 - Deploy a ReactJs web application

1. Create a ReactJs application and test it in locally
2. Once you are ready to create a docker image you will need to include two new files:
   1. Dockerfile
   2. .dockerignore
   3. docker-compose.yml
3. Generate the image build with the docker's command:
   1. docker build -t react-poc-app .
4. Test it in locally (in our case, the ReactJs application is configured to run in the port 3000):
   1. docker run -d -p 3000:3000 --name reactapp react-poc-app
5. **Important**: Due this PoC was generated after the PoC#1, we didn't need to create the environment again. If is not your case, you will need to execute the steps from 6th to 21th described in PoC#1.
6. Tag our local image with a "development" tag on it:
   1. docker tag react-poc-app [ciamcontaineracr.azurecr.io/react-poc-app:development](http://ciamcontaineracr.azurecr.io/react-poc-app:development)
7. Push the new tagged image into our local repository
   1. docker push [ciamcontaineracr.azurecr.io/react-poc-app:development](http://ciamcontaineracr.azurecr.io/react-poc-app:development)
8. Create a new secret into our Kubernetes for our application:
   1. kubectl create secret docker-registry **ciamcontaineracrreactpocapp** --docker-server [ciamcontaineracr.azurecr.io](http://ciamcontaineracr.azurecr.io/) --docker-email [iban.balasch@es.nestle.com](mailto:iban.balasch@es.nestle.com) --docker-username=ciamcontaineracr --docker-password "/G6zJp3gUaTXQNePretKehVfdB8F7TRu"
   2. The required steps needed to obtain the docker password are described in the step 21 in the PoC#1
9. Create a Kubernetes's POD using our ReactJs Docker image (from ACS):
   1. kubectl create -f react.yaml
10. Wait until POD is created and the service is exposed and assigned a public IP.
    1. To know the progression, you can run the command kubectl get service --watch
11. In a browser window, navigate to the public IP assigned to our ReactJs web application to test that all works (published at http://www,ciampoc.tk)

[[](https://dsu-confluence.nestle.biz/download/attachments/38319388/.dockerignore?version=1&modificationDate=1526570960000&api=v2).dockerignore](https://dsu-confluence.nestle.biz/download/attachments/38319388/.dockerignore?version=1&modificationDate=1526570960000&api=v2)[[](https://dsu-confluence.nestle.biz/download/attachments/38319388/Dockerfile?version=1&modificationDate=1526570961000&api=v2)Dockerfile](https://dsu-confluence.nestle.biz/download/attachments/38319388/Dockerfile?version=1&modificationDate=1526570961000&api=v2)[[](https://dsu-confluence.nestle.biz/download/attachments/38319388/docker-compose.yml?version=1&modificationDate=1526991435000&api=v2)docker-compose.yml](https://dsu-confluence.nestle.biz/download/attachments/38319388/docker-compose.yml?version=1&modificationDate=1526991435000&api=v2)

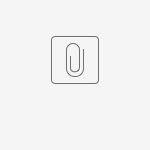


Like [Wikipedia](https://en.wikipedia.org/wiki/OSI_model) explains, the Open Systems Interconnection model (OSI model) is a conceptual model that characterizes and standardizes the communication functions of a telecommunication or computing system without regard to its underlying internal structure and technology. Its goal is the interoperability of diverse communication systems with standard protocols. The model partitions a communication system into abstraction layers. The original version of the model defined seven layers. A layer serves the layer above it and is served by the layer below it. For example, a layer that provides error-free communications across a network provides the path needed by applications above it, while it calls the next lower layer to send and receive packets that comprise the contents of that path. Two instances at the same layer are visualized as connected by a horizontal connection in that layer.

In our case, Kubernetes service only reach at layer 4, the transport layer. That means that there are no options available to inspect the traffic, all is redirected to the specified port. If we want to reach the layer 7 we need to implement an external solution called [Ingress](https://docs.microsoft.com/en-us/azure/aks/ingress). To do that we need to follow the next steps:

1. First of all, as a reminder, the installation of Ingress is per cluster, if we configure Ingress in our cluster, the rest of the deployments must work with it. Otherwise, if we don't want to use Ingress in the applications X and Y (for example) and want to use a simple Kubernetes's service we will need to create a different cluster that the Ingress's cluster.
2. [Choco](https://chocolatey.org/) is available by default in Kubernetes so we will use it to install Ingress with the command: **choco install kubernetes-helm**
3. Update repository with the latest updates: **helm repo update**
4. Initialize Ingress to be deployed in our cluster: **helm init**
5. Install it in our cluster: **helm install stable/nginx-ingress --namespace kube-system --set rbac.create=false --set rbac.createRole=false --set rbac.createClusterRole=false**
6. Ingress is installed in another namespace so, to see the public endpoint assigned to Ingress we need to specify the namespace in the syntax: **kubectl get svc -n kube-system**
7. The next and last step is to create a new Deployment into Kuberntes with a Yaml that must specify that works with Ingress and the configuration related to it (see file attached called "reactWithIngress.yaml")

**Important**: We will need to use calico (implements a security networking layer) with Kubernetes and Ingress. More info at <https://docs.projectcalico.org/v3.0/getting-started/kubernetes/tutorials/using-calicoctl>

[[](https://dsu-confluence.nestle.biz/download/attachments/38319388/reactWithIngress.yaml?version=1&modificationDate=1526983251000&api=v2)reactWithIngress.yaml](https://dsu-confluence.nestle.biz/download/attachments/38319388/reactWithIngress.yaml?version=1&modificationDate=1526983251000&api=v2)

The YAML file, to use with Ingress, can be defined to include subdomains, redirects or everything related with layer 7. For example, this is a YAML definition for a subdomains host:

spec:

rules:

- host: foobar.com

http:

paths:

- backend:

serviceName: foobar

servicePort: 80

- host: api.foobar.com

http:

paths:

- backend:

serviceName: foobar

servicePort: 80

- host: admin.foobar.com

http:

paths:

- backend:

serviceName: foobar

servicePort: 80

- host: status.foobar.com

http:

paths:

- backend:

serviceName: foobar

servicePort: 80

More info at <https://github.com/nginxinc/kubernetes-ingress>

**Important**:

If we want to use Ingress we will need a hostname already defined, meanwhile we are waiting the "administrative process" related, we can use a free namespace from [Freenom](http://www.freenom.com/) to assign a temporal hostname.

Helm is the Kubernetes package manager ([https://helm.sh](https://helm.sh/)).



There are some tips and tricks found when we were working on this PoC and is good thing to know. Let's go!

**Want to open a bash shell for any deployed pod?**

kubectl exec -it <pod\_name> -- /bin/bash

For example, this tip is useful to execute curl  to get information about our exposed service using the internal IP, otherwise is not possible.

**I opened a bash shell but I cannot use curl!!!**

apt-get update -y && apt-get install curl to install curl

In your connected bash you will need to install it first

**I want to get more information about my PODs**

kubectl get pods -o wide

The option "-o wide" is available for any kubectl command.

**I create a new VM for Jenkins, now what?**

First of all, we need to retrieve the required data to login into it so, we will execute the next steps:

Login using a valid account into AAD specifiying the subscription Id (on this case, "CIAM - DEV - Core tools":

Login-AzureRmAccount -SubscriptionId b0122ca4-b7a9-4e33-9cdb-fa965a7849a0

Get the value of the secret key stored in our VM (on this case, the VM's name is "CIAMJenkins")

az vm run-command invoke --resource-group CIAMJenkinsResourceGroup --name CIAMJenkins --command-id RunShellScript --scripts "cat  /  etc /  ssh  /  ssh\_host\_ecdsa\_key.  pub"

This will return a JSON object and, inside the property "message" you will find the value of the ECDSA key. We will need to modify our known\_hosts file (located in c:\Users\<our\_user>\.ssh\known\_hosts) and include it.

Once you will followed all these steps, you can access to the Jenkins instance creating a local tunnel to it

ssh -L 127.0.0.1:8080:localhost:8080 [ciamjenkins@jenkinsciam.westeurope.cloudapp.azure.com](mailto:ciamjenkins@jenkinsciam.westeurope.cloudapp.azure.com)

**Where is the Kubernetes's dashboard?**

az aks browse --resource-group CIAMContainerTest --name CIAMCluster

You will need to execute this command to create a tunneling between Kubernetes and our local machine. Seems that in MacOS is not working properly but in Windows works so...

**Do we must create a HELM package?**

Yes, use HELM to create packages. Must be stored in a repository and install it like "helm install <my\_package>". The other way is use ARM + scripts to create them. The last way is use DRAFT (streamlined kubernetes development)

**Typical or Simplify flow?**

The typical flow, using Helm, is something like this:

Code →  Build  → Helm  (create package)→  Test

                  |       /

                  V   /

            Create docker image into repository ACS

The Simplify flow (is not CI/CD, is internal developer workflow) is like this:

Code → Draft (create dockerfile and a helm chart for service composition)  → Deploy container → Test

Draft URL: <https://github.com/Azure/draft>

**The most Kubectl's command used**

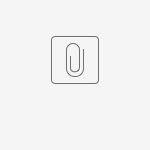
| **Command** | **Description** |
| --- | --- |
| kubectl get pods | List the current pods available in our current cluster |
| kubectl get deployments | List the current developments |
| kubectl get services | List the current services |
| kubectl get pods | List the current pods |
| kubectl logs <image\_name> | Show the existant logs related to the image name specified |
| kubectl delete pod --all | Delete all the pods from Kubernetes |
| kubectl expose deployment/<deployment\_tag> --type="NodePort" --port 8081 | Expose a service into port 8081 for the development specified |
| kubectl get secret | Show all the secrets created in our current cluster |
| kubectl get secret <secret\_name> --output=yaml | Display information related to the secret specified |
| kubectl api-versions | Show all the API versions supported by our Kubernetes's instance |
| kubectl get sc | Get the storage classes in our current cluster |
| kubectl get services --namespace kube-system | Get services from another namespace (like Ingress) |
| kubectl apply -f <file.yaml> | If we have something created using a YAML file, if we want to made changes, instead run create command again, use this command to reflect the changes specified in the YAML. |
|  |  |



PoC #1 - Nexus service

This is the YAML file used to deploy a Nexus Docker image as a service into Kubernetes. Will divide the explanation in three parts, each about one kind:

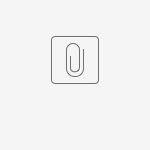
|  |  |
| --- | --- |
| POD section | apiVersion: v1 kind: Pod metadata:   name: nexus   labels:     app: nexus spec:   containers:     - name: nexus       image: [ciamcontaineracr.azurecr.io/nexus:v1](http://ciamcontaineracr.azurecr.io/nexus:v1)       ports:         - containerPort: 8081   imagePullSecrets:     - name: ciamcontaineracrnexus --- |
| Service section | apiVersion: v1 kind: Service metadata:   name: nexus-deployment spec:   replicas: 1   template:     metadata:        labels:         app: nexus-deployment     spec:       containers:       - name: nexus         image: [ciamcontaineracr.azurecr.io/nexus:v1](http://ciamcontaineracr.azurecr.io/nexus:v1)       imagePullSecrets:         - name: ciamcontaineracrnexus --- |
| LoadBalancer section | apiVersion: v1 kind: Service metadata:   name: nexus-service spec:   type: LoadBalancer   selector:     app: nexus   ports:   - protocol: TCP     port: 80     targetPort: 8081     --- |

[[](https://dsu-confluence.nestle.biz/download/attachments/38319388/nexus.yaml?version=2&modificationDate=1538576170000&api=v2)nexus.yaml](https://dsu-confluence.nestle.biz/download/attachments/38319388/nexus.yaml?version=2&modificationDate=1538576170000&api=v2)

PoC #2 - ReactJs web application

The YAML file used is different than the used in the PoC#1 due, instead define a service, we define a "Deployment" and "LoadBalancer" entries. The idea to use a "Deployment" instead a POD is to give more flexibility to our cluster's definition if we need add new service due balance or performance issues.

|  |  |
| --- | --- |
| Deployment section | apiVersion: apps/v1 kind: Deployment metadata: name: reactpocapp spec: selector: matchLabels: app: reactpocapp replicas: 1 template: metadata:  labels: app: reactpocapp spec: containers: - name: reactpocapp image: [ciamcontaineracr.azurecr.io/react-poc-app:development](http://ciamcontaineracr.azurecr.io/react-poc-app:development) imagePullSecrets: - name: ciamcontaineracrreactpocapp |
| LoadBalancer section | apiVersion: v1 kind: Service metadata: name: reactpocapp-service spec: type: LoadBalancer selector: app: reactpocapp ports: - protocol: TCP port: 80 targetPort: 3000 |

[[](https://dsu-confluence.nestle.biz/download/attachments/38319388/react.yaml?version=3&modificationDate=1526990641000&api=v2)react.yaml](https://dsu-confluence.nestle.biz/download/attachments/38319388/react.yaml?version=3&modificationDate=1526990641000&api=v2)

Others definitions

The Persistant volume is something important to persist data into a storage volume, otherwise the data is removed each time that the POD is putting down.

|  |  |
| --- | --- |
| Storage class  This kind of storage define a storage class to be used by another PODs as a persistant volume | kind: StorageClass apiVersion: [storage.k8s.io/v1](http://storage.k8s.io/v1) metada: name: azurefile provisioner: [kubernetes.io/azure-file](http://kubernetes.io/azure-file) mountoptions: - dir\_mode=0777 - file\_mode=0777 - uid=1000 - gid=1000 parameteres: skuName: Standard\_LRS |
| PersistantVolumeClaim  The best practice is to use a claim to access at the persistant volume created in another thread.  Is a bad practice that the Deployment points directly to the persistant volume | kind: PersistentVolumeClaim apiVersion: v1 metadata: name: myclaim spec: accessModes: - ReadWriteOnce volumeMode: Filesystem resources: requests: storage: 8Gi storageClassName: slow selector: matchLabels: release: "stable" matchExpressions: - {key: environment, operator: In, values: [dev, test, ..](https://dsu-confluence.nestle.biz/pages/createpage.action?spaceKey=DCIAM&title=dev%2C+test%2C+..&linkCreation=true&fromPageId=38319388)]} |

More info at <https://kubernetes.io/docs/concepts/storage/persistent-volumes/>



We deploy a simple ReactJs application in the folder "[POC\_Jenkins\_CI](https://dsu-bitbucket.nestle.biz/projects/CIAM/repos/ciam_development/browse/POC_Jenkins_CI)" with this structure:

src/

├── \*.js # React components

├── \*.test.js # React component tests

├── actions.js # Redux actions

└── reducer.js # Redux reducer

As a requirement, we include the readme.MD to describe structure, purpose and tech-stack used into.

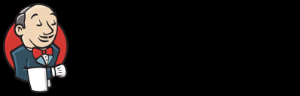
In the "CIAM\_Development" repository was initialized to use git flow, due that you will find the branches master and development already created and, of course, you can "surf" inside development branch and see some features branches merged into for test purpose.

Due AAD authentication, is a requirement to include an authorization header when you try to pull or push data from/to Bitbucket

git config --global http.extraheader "Authorization: Bearer MjM2NDQ4MTI0NzQwOs44BrjECDedegMwTSCcnc41LhmE"

**WIP**

* Connect Bitbucket with Jenkins



In this PoC, Jenkins is deployed as an application service in Azure via virtual machine. Is important to remark it because in the final implementation we will choose between this option or move to be exposed via Kubernetes.

The URL to this poc-service is [http://jenkinsciam.westus.cloudapp.azure.com](http://jenkinsciam.westus.cloudapp.azure.com/)  (<https://bit.ly/2rx0mSF>)

* At this moment, 16th May'18, this virtual machine is stopped due blocks related with security,
* If you need more information about status, you can contact with [Balasch,Iban,BARCELONA,CIAM](https://dsu-confluence.nestle.biz/display/~iban.balasch@es.nestle.com).

To create a tunnel between Jenkins in Azure and your local machine (and you will access to Jenkins at [http://localhost:8080](http://localhost:8080/)), type this in a terminal console:

ssh -L 127.0.0.1:8080:localhost:8080 [ciamjenkins@jenkinsciam.westeurope.cloudapp.azure.com](mailto:ciamjenkins@jenkinsciam.westeurope.cloudapp.azure.com)

**Plugins installed**:

* <https://plugins.jenkins.io/azure-acs> (ACS)

**Jenkins credentials**:

Subscription Id  b0122ca4-b7a9-4e33-9cdb-fa965a7849a0

Client Id: cb5a0bbd-905c-4bb4-b4b0-c630f20d5a5f

Client secret: ---

Tenant Id: 12a3af23-a769-4654-847f-958f3d479f4a

**WIP**: Jenkins will use Azure Container Instances (ACI) to wake up a virtual machine, launch a job to build-deploy-whatever and then turn off when finish it

**WIP**: Follow the steps defined at <https://docs.microsoft.com/es-es/azure/aks/jenkins-continuous-deployment> to create a CI/CD with Jenkins and AKS.

**WIP**: Create a Jenkins's shell script like <https://raw.githubusercontent.com/Azure-Samples/azure-voting-app-redis/master/jenkins-tutorial/deploy-jenkins-vm.sh> and <https://raw.githubusercontent.com/Azure-Samples/azure-voting-app-redis/master/jenkins-tutorial/config-jenkins.sh>

**Azure Active Directory Application Registry:**

Application Id: cb5a0bbd-905c-4bb4-b4b0-c630f20d5a5f

Client Id: 77a2541d-8cea-47c2-8b6f-38669759b287

Client secret: GobXJdvv+8EqMlJPLImJyHkY5Ql51CtBuiYdk9Z5m2E=

Another client secret: DPOzdHxYkKXAUABOubj0rHFNbGk+d4eD0d2hLrlIrz0=

Tenant ID: 12a3af23-a769-4654-847f-958f3d479f4a

**WIP**

* Waiting for grant permissions to access at Azure Active Directory to Jenkins application
* Connect Jenkins with Bitbucket
* Create all the required process to accomplish the requirements described in the [high-level architecture diagram](https://dsu-confluence.nestle.biz/pages/viewpage.action?pageId=38318332).

**Mandatory reading**

* <https://docs.microsoft.com/en-us/azure/architecture/reference-architectures/jenkins/>



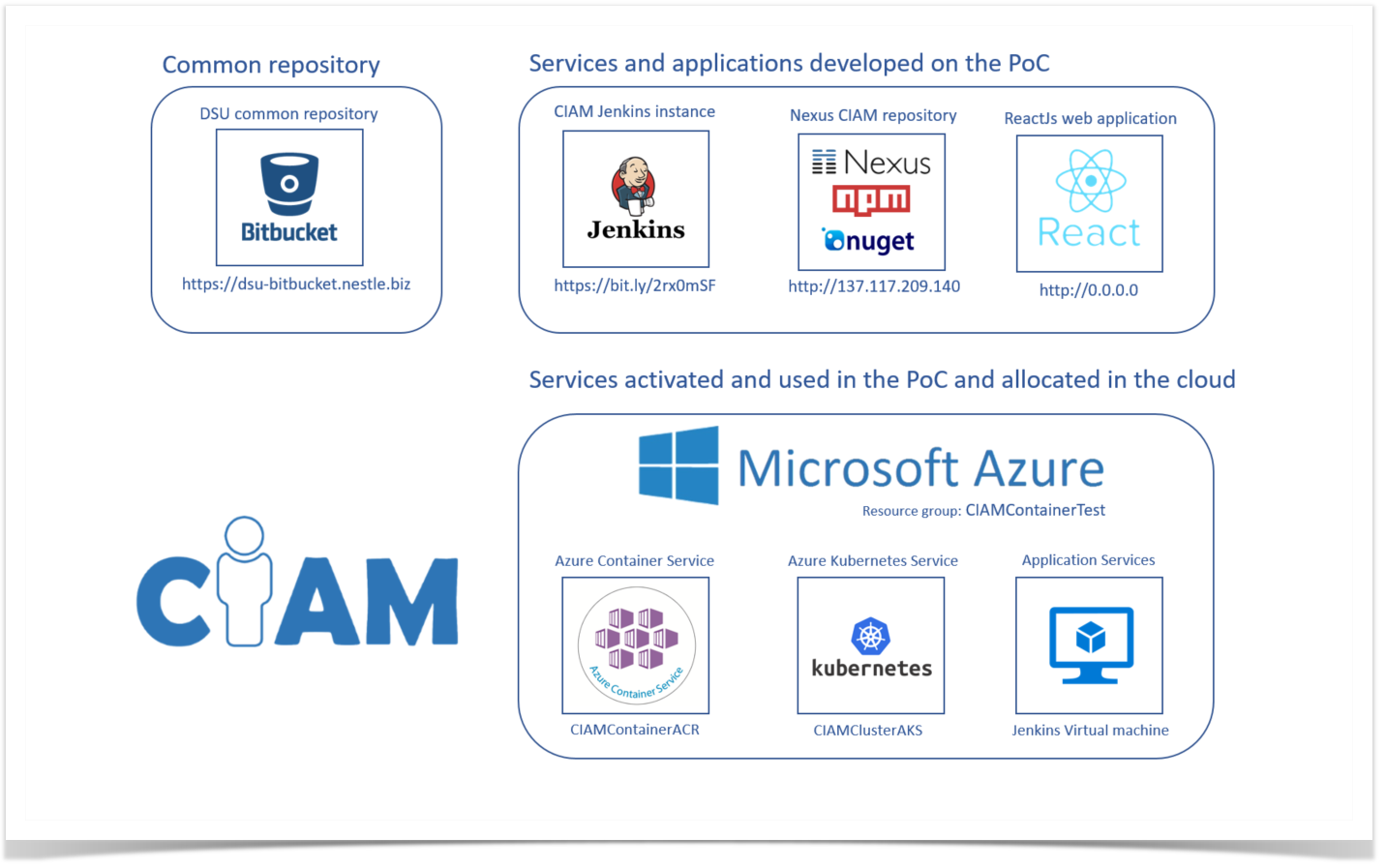
To access at the Microsoft Operation Management suite go to [https://ciamworkspce.portal.mms.microsoft.com](https://ciamworkspce.portal.mms.microsoft.com/)

To access at the Azure Log Analytics, linked to our resources, go to <https://portal.loganalytics.io/subscriptions/b0122ca4-b7a9-4e33-9cdb-fa965a7849a0/resourcegroups/ciamcontainertest/workspaces/ciamworkspace>

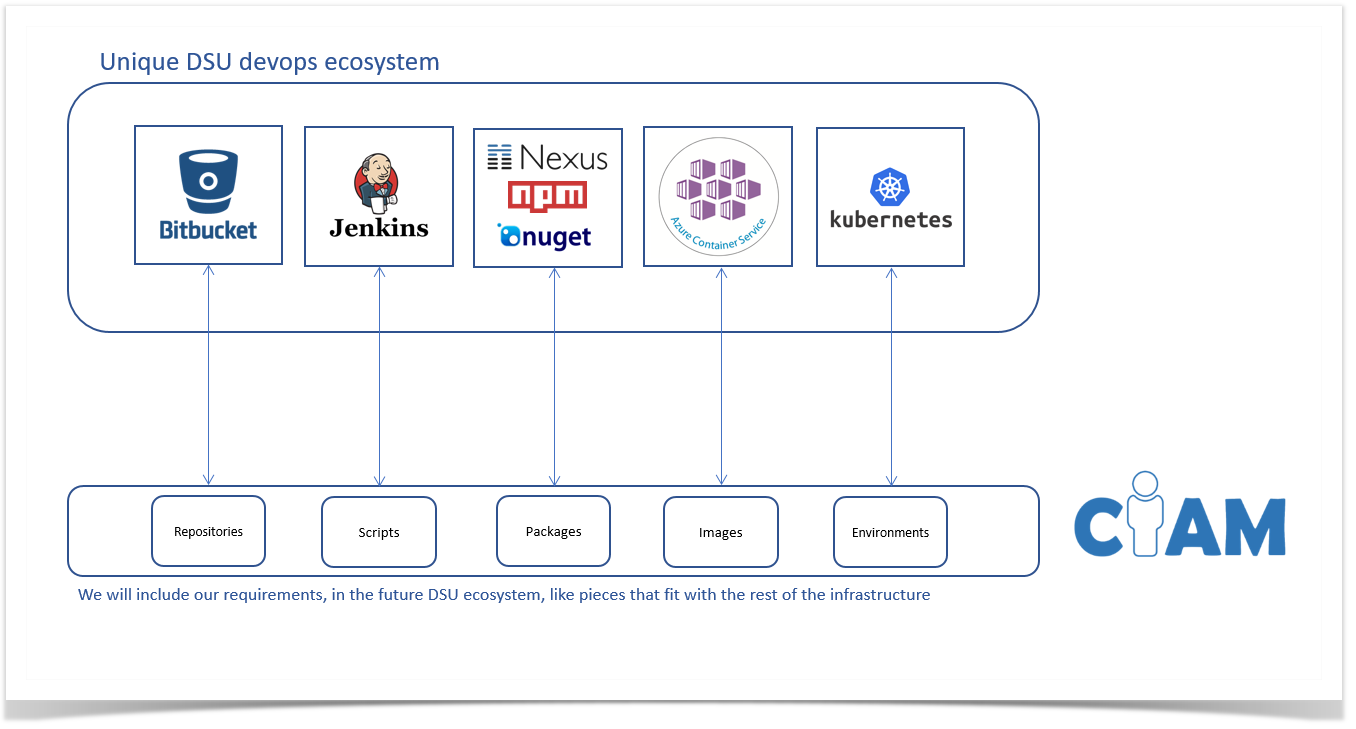
* To enable analytics, follow the steps defined in <https://docs.microsoft.com/en-gb/azure/monitoring/monitoring-container-health#enable-container-health-monitoring-for-existing-managed-clusters>

The alerts configured at available at: <https://portal.azure.com/#@nestle.onmicrosoft.com/resource/subscriptions/b0122ca4-b7a9-4e33-9cdb-fa965a7849a0/resourcegroups/mms-weu/providers/Microsoft.OperationalInsights/workspaces/CIAMWorkspce/alertsV2>

The infrastructure used/created related with this PoC are defined on this schema:



But, in the future, this structure will be change to fit with the DSU devops's vision. All will be centralized and we will fit our pieces in the DSU structure:





Nexus repository was deployed as a Docker's image in ACS. No extra configuration was included, was used the default ones. The only parametrization was to remove the anonymous access and create a guest user.

To access at this instance, the URL is [http://137.117.209.140](http://137.117.209.140/). You can take a look as a anonymous guest using the credentials:

* user: guest
* password: [c](http://137.117.209.140/)iamtest2018

To access to the npm repository from your local machine, you will need to include a new npm registry to point to Nexus. The syntax is:

npm config set registry [http://137.117.209.140](http://137.117.209.140/)

In case that you want to include a NuGet artifact, you will need to include it using the package manager console and type:

nuget add <package\_name>.nupkg -source http://137.117.209.140

**WIP**

* Create a NuGet and npm repository (was already created and tested -works!- in another deployment but not in the current one)



First of all, the Azure CLI tools are required to interact with Azure, you can install it from <https://docs.microsoft.com/es-es/cli/azure/install-azure-cli>. Once installed, you can run the required commands from your Powershell terminal windows.

To access at the Kubernetes's dashboard, you will need to open a powershell terminal window and type this command:

az aks browse --resource-group CIAMContainerTest --name CIAMCluster

This will open a tunnel between AKS and your machine and expose the dashboard in your localhost (by default, port 8001, keep in mind if you have another service running at that port).

If we want to create a template, we can use the wizard going to the option "Create kubernetes using "Add" wizard"

**Important**: Remember to create at least  two service principals, one for development-tests-preprod environments and another for production environments (ACS)