Innovation Beta : Container aaS   
 v0.4, avec liens sur Vidéos et new Logo

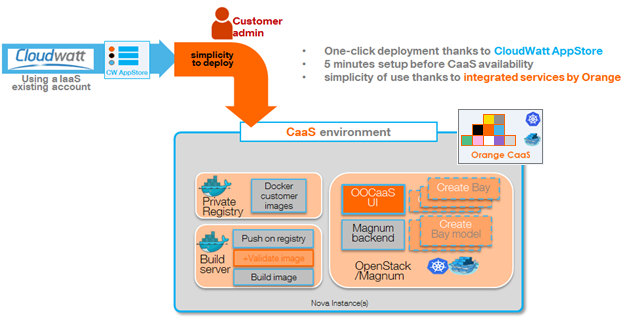


**Context**

’Container as a Service’ (CaaS) is a Beta service that will deliver all the necessary material to execute **dockerized services** inside OpenStack KVM environment that provides flexibility thanks to the CloudWatt IaaS, security thanks to OpenStack virtualized network, and simplicity thanks to the content of this bundle. Please follow the [why-devops-with-dockerized-micro-services](https://www.dailymotion.com/video/x4cinix_caas-teasing-why-devops-with-dockerized-micro-services_tech) video to understand the DevOps move!

**The promises are**

* One-Click deploy & 5 minutes: you have your **Docker tm-based ContainerAAS** infrastructure.
* One click & 3 minutes: you have your first Container cluster (*Magnum Bay*) fitted with Docker **Swarm** tm or Google tm **Kubernetes** tm **COE** (*Container Orchestration Engine*).
* Few clicks: you integrate CaaS inside your existing **Jenkins chain** in order to execute **DevOps**



CaaS is an easy-to-deploy bundle, available from the CloudWatt Application store.

Three major components are delivered:

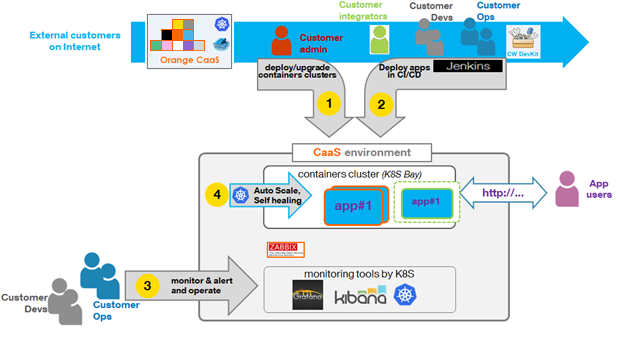
* A KVM instance that provides a ‘Private Docker registry’ to store your future Docker images
* A KVM instance that provides a preconfigured ‘Build server’ to build your future Docker images
* A KVM instance that provides the CaaS Backend (available via API) and User Interface, available via HTTP.

More info and tutos are available on <https://www.cloudwatt.com/fr/labs/caas.html>

Videos of the service are available on <https://www.dailymotion.com/Cloudwatt> .

Thanks to this CaaS environment, you will benefit from the OpenStack/Magnum project that encapsulates the management of Docker tm containers clusters for Kubernetes tm or Docker Swarm tm. A ‘Cluster’ (’*Magnum* ***Bay***’) is a set of one ‘*Master*’ KVM instance that delivers the API and UI for the selected Docker COE (Container Orchestrator Engine) plus one or several ‘*Nodes*’ KVM instance(s) that host the customer containers.

If you select Kubernetes tm cluster as COE when creating a ‘*Magnum* ***Bay Model***’, some complementary tools will be available: 1) embedded Kubernetes Dashboard to track the running containers, 2) embedded ELK (ELastic Search / Kibana) logging system and 3) ‘Collect’D / Graphana analytics.



As ‘Customer Admin’, after the ‘One-Click’ deployment of the CaaS bundle in your existing OpenStack tenant, you will:

* Allocate your first ‘BayModel’ and your ‘Bay’ :see ‘1’.
* Then configure your DevOps tooling (see Jenkins with an example of Ansible ‘cookbook’ provided by CaaS): see ‘2’.

Then for every Dockerized application, the customer DevOps team will develop and deploy the application with CaaS according to various roles: see ‘2’:

* As **customer senior developer**, you will define the descriptors of the dockerized application:
  + The classical POM.XML file used by Jenkins to build your application from the source code version controlled in Git.
  + The ‘*docker file*’ that list the content of your future Docker images
  + The *YML COE descriptor*, so that the selected Kubernetes or Swarm orchestrator will understand how to deploy or update your application inside a Bay
* As any **customer developer**, you will modify the source code of the application and run the ‘**All in one’ Jenkins job** that will execute the following steps for ‘non production-grade’ environment:
  + Build the application elements: in our ‘PetClinic’ example you will find a Java Spring application that will build a classic WAR file.
  + Build the Docker image(s): in our ‘PetClinic’ example, you will discover that we build a Docker image made of the ‘latest’ tomcat available on the Docker Git Hub and the ‘PetClinic’ War file and very few params
  + Deploy the application inside a Bay.
* As any **customer integrator**, you will optionally adapt the Jenkins job in order to benefit from a partial job that will only deploy an application using either the latest generated ‘PetClinic’ image or a previous one to test non regressions and new features ☺
* As any **customer OPerationS member**, you will optionally duplicate the ‘All in one’ job in order to define your own process for ‘production-grade’ environment and update the enterprise ‘Change’ database.

As any customer DevOps team member (and especially the OPS members) you will benefit from the Kubernetes add-ons (*Logging, metering and dashboarding*) and CaaS auto-monitoring (*with Zabbix-based monitoring for CaaS elements and optionally the customer’s containers*): see ‘3’.

As customer team member, you will benefit from the automatic Kubernetes features for auto-scale Up&Down of your containers (if detailed inside your application’s Kubernetes descriptor) as well as auto-repair == ‘*Slef-healing’*): see ‘4’.

**Preparation**

**The versions**

* Magnum 1.1
* Docker v1.10
* Kubernetes v1.2.2
* Swarm 1.1.3

**The prerequisites to deploy this stack**

These should be routine by now:

* Internet access
* CloudWatt credentials and a valid KeyPair for the future KVM instances
* The knowledge on how to use the CW AppStore: let’s one-click on ‘**deploy’** ☺

**Size of the instances**

The ‘One-Click’ bundle is packaged as an OpenStack Heat stack. By default, the Three CaaS instances will be allocated with the ‘m1.small’ flavor. We recommend you not to minor this flavor’s size.

Per instance, a Cinder volume is attached in order to keep the persistent data.

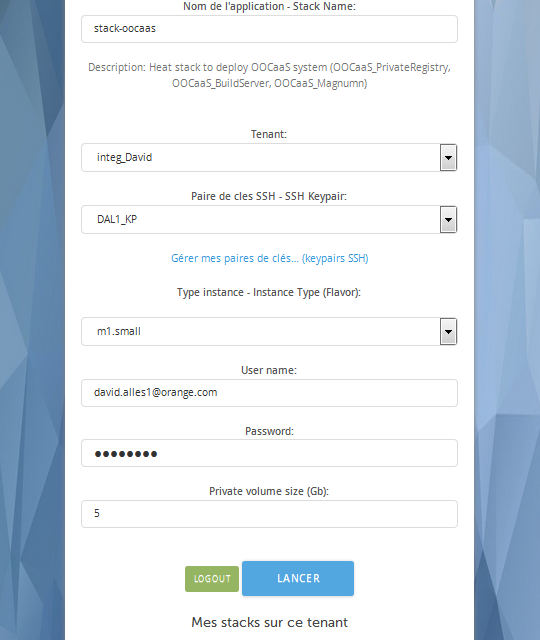
**What will you find in the repository**

Once you have cloned the github from [/cloudwatt/applications/application-caas](https://github.com/cloudwatt/applications/tree/master/application-ocaas) **<Flo, verifie>** , you will find in the bundle-CaaS/ repository:

* **application-caas\_beta\_1.0.heat.yml**: HEAT orchestration template. It will be use to deploy the necessary infrastructure.
* **PetClinic\_sample.zip**: Java Spring example, to build and deploy a Tomcat-based Docker image and use a existing mySql Docker image
* **Tweet\_sample.zip**: Cloud Native application as a composite of existing Docker images
* **README.md** and **README-EN.md** (this current document)
* **CaaS\_howToTroubleshoot.pdf:** future document.

**Install CaaS in 1-click procedure:** *See* [*one-click-caas-deployment*](https://www.dailymotion.com/video/x4cinqb_one-click-caas-deployment_tech) *video*

CaaS start with the 1-click of Cloudwatt via the web page [Apps page](https://www.cloudwatt.com/fr/applications/) on the Cloudwatt website.   
Choose CaaS app, press DEPLOY. After entering your login / password to your account, the wizard appears:

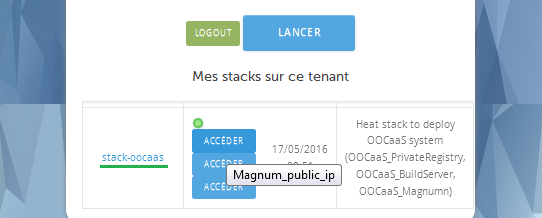


As you may have noticed the 1-Click wizard asked to reenter your password Openstack.  
By default, the wizard selects the flavor “m1.small”. A variety of other instance types exist to suit your various needs, allowing you to pay only for the services you need. Instances are charged by the minute and capped at their monthly price (you can find more details on the [Pricing page](https://www.cloudwatt.com/en/pricing.html) on the Cloudwatt website).

Please remember that you are providing your KeyPair that will be used in order to postConfigure the future three KVM instances: this will be your way to SSH on those instances for troubleshooting or granting your colleagues on them if required.

**Press DEPLOY**. The 1-click handles the launch of an Heat stack that triggers the allocation of three instances and the related OpenStack elements (cinder volumes, neutron internal private network…)

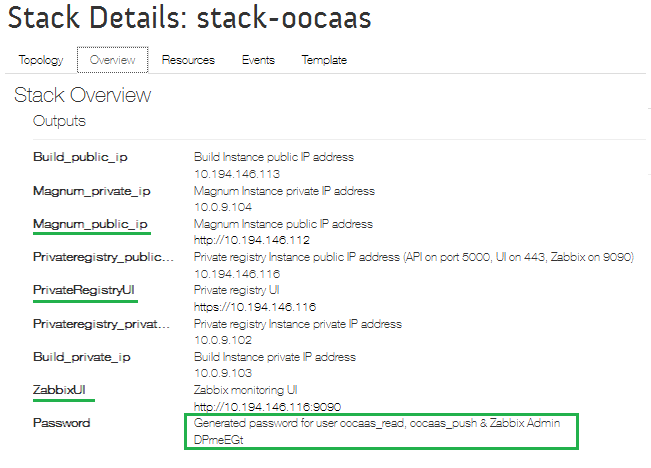
You can see its progression by clicking on its name which will take you to the Horizon console.   
When all modules become “green”, the creation is finished.



You can then find three URLs that are accessible via 3 floatingIps allocated by the 1-click:

* ‘Magnum\_public\_ip’, which provides the access to the CaaS portal, Magnum\_UI-based.
* ‘PrivateRegistry UI’, to be used with the appropriated certificate and related login/ Password see below.
* ‘Zabbix UI’, to be used to access to the embedded self-monitoring tool with the related login/ Password see below.

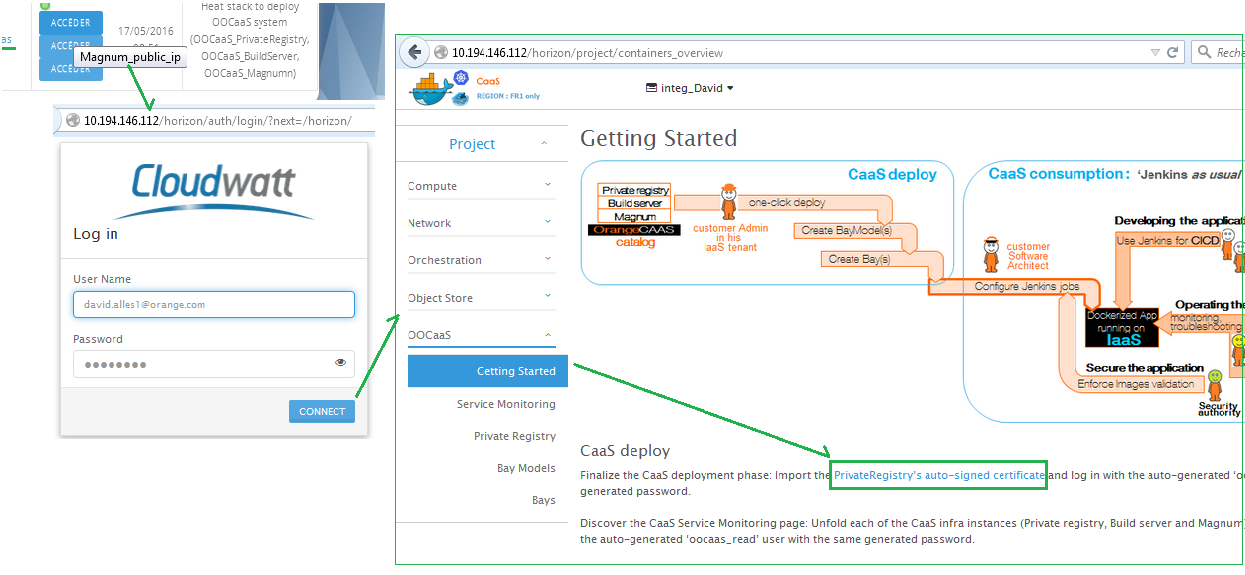
In the standard Horizon/orchestration console, you can dive inside the Stack outputs.   
You will retrieve the previous 3 URLs as well as the generated password that will be used during the end of CaaS setup.



**… Keep in mind this stack’s auto-generated password!!!**

**Finish the CaaS setup:** *Second part of the* [*one-click-caas-deployment*](https://www.dailymotion.com/video/x4cinqb_one-click-caas-deployment_tech) *video*

As detailed in the **video named ‘One-Click deploy’**, please: follow the ‘*Getting started*’ procedure after the authentication of the new CaaS portal



Finishing the setup of the CaaS infrastructure is simple:

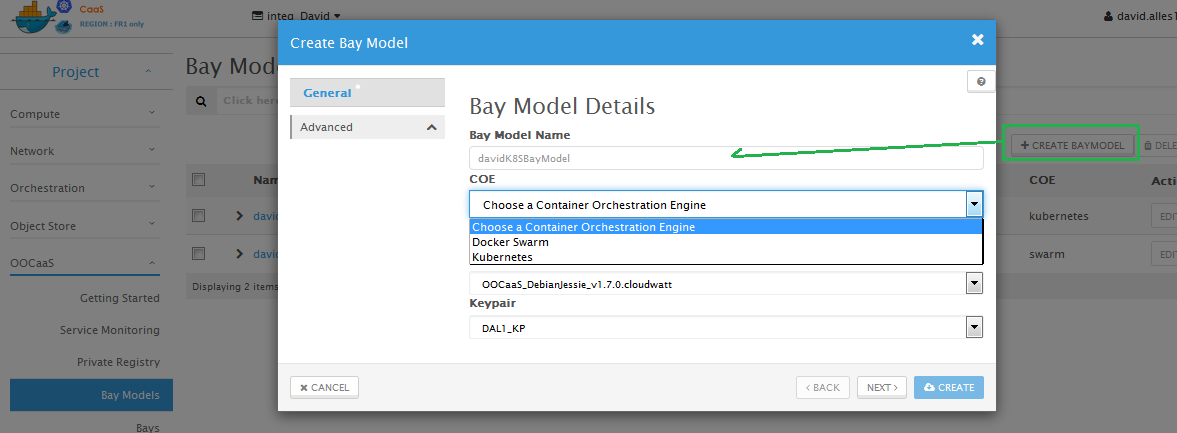
1. Access to the Magnum URL in order to login. Use your standard CW credentials as this new ‘Over the IaaS’ service is connected to CloudWatt authentication ‘Keystone’ service.
2. On the ‘*CaaS/Getting’* started page, please click on the ‘PrivateRegistry’s auto-signed certificate’ link in order to accept the auto-signed HTTPS certificate, then login on the related page with the following credentials: login=’*oocaas\_read*’, pwd=<**StackAutoGeneratedPassword**>

You have your CaaS infrastructure

**Creating your first Container clusters**

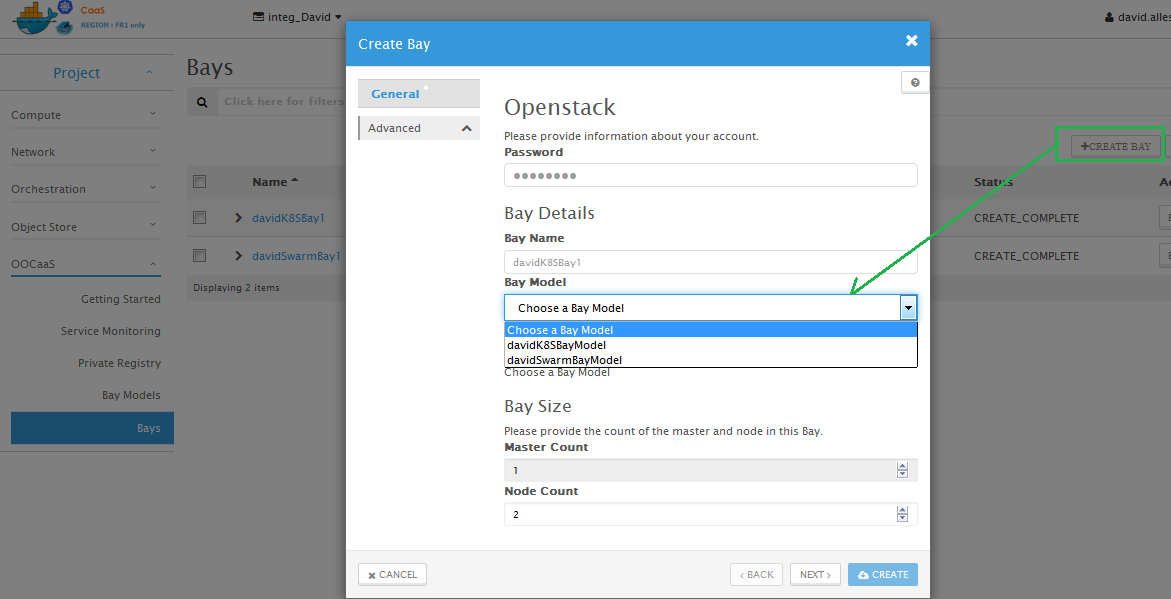
**Creating BayModel:** See [managing-baymodels](https://www.dailymotion.com/video/x4cinwm_caas-managing-baymodels_tech) video.

Once setup, this CaaS infrastructure allows you to allocate your first ‘BayModel’ to identify your Docker cluster’s template(s) with default parameters: just give a name and select either ‘Kubernetes’ or ‘Swarm’ COE.



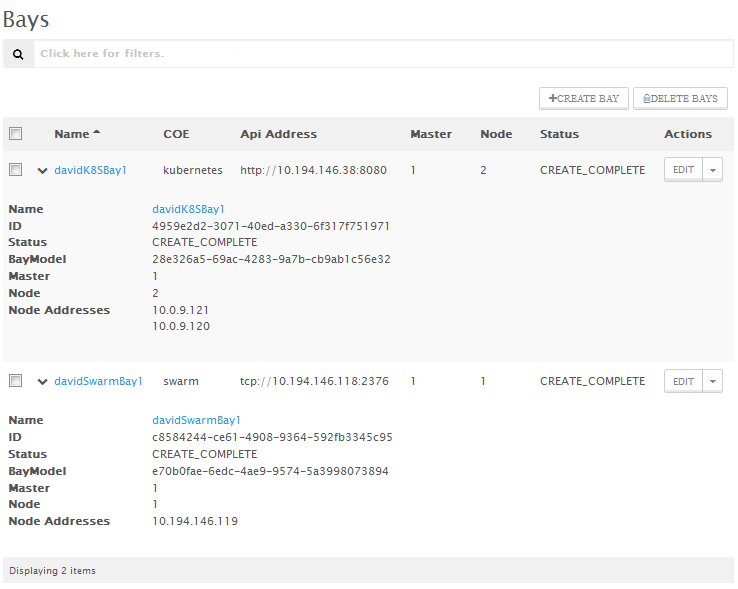
**Creating your first Bay(s):** See [creating-and-updating-swarm-bay](https://www.dailymotion.com/video/x4cisjo_caas-creating-and-updating-swarm-bay_tech) and [creating-and-updating-k8s-bay](https://www.dailymotion.com/video/x4b0bii_creating-and-updating-k8s-bay_tech) videos.

Once you have your BayModel(s), then simply create one or mode bays.



* This step will drive the launch of an OpenStack/Heat stack: please provide your password
* Give a name to the bay, select the BayModel, select the number of ‘Nodes’ that are the KVM instances that will host your customer containers running your Dockerized application.
* When ‘Create’, then a Heat Stack is launched. Please wait for the completion of the creation and review the resulting clusters.

As a result, one cluster of each COE will be similar to the following screenshot



* Status is completed (meaning Heat stack deployment is OK and the COE related postConfiguration is OK too (*synchronization between the ‘Master’ and its ‘Nodes’*)
* A single master is allocated (*in the Beta version, no possibility to multi-instanciated this, even if Magnum is capable for; in the future version this High Availability feature will be added*)
* As your choice, one or mode ‘Nodes’ will be allocated. In the K8S (Kubernates, guys) naming those are ‘*Minions’.*

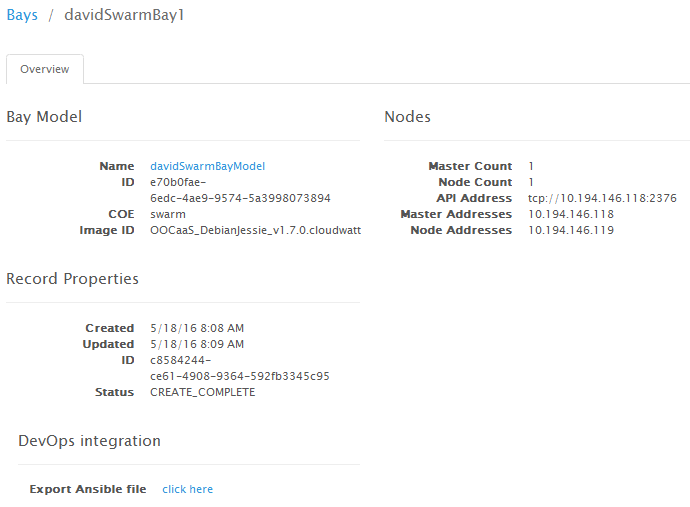
You now have your context in order to deploy your dockerized micro services

**Understanding the possibilities, depending on COEs: ‘Swarm is simple’ vs ‘Kubernetes is features rich’**

**Swarm Bay**

Docker Swarm technology in this v1.1.3 version is simple and easy to use (*see the following chapter on how to deploy PetClinic sample with it, or the related video*).

This COE provides a ‘Docker Compose compatible YML file to describe the Dockerized application. **Swarm drives the deployment of the containers**: That’s (*only*) it!



In CaaS user Interface, the Swarm-based bay is displayed with few information:

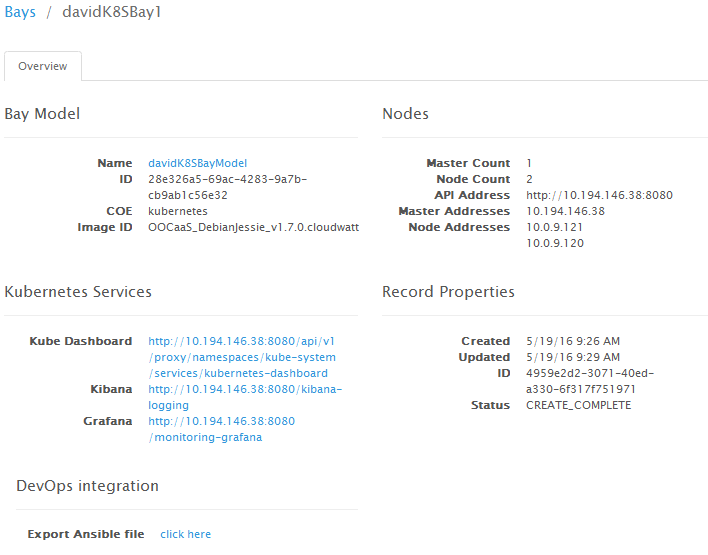
* You have the API address of the Swarm API service (hosted on the ‘*Master’* instance).
* You must dig into the nodes in order to discover which one is run which deploy container
* You must expose your container on the external world via OpenStack/Neutron network features (Load balancer and/or FloatingIp)
* You can allocate an OpenStack/Cinder volume and give it to one containe (as an example, the ‘*mySql’* database container in order to persist the data )

In one work: Simple: YOU DO the job!

**Kubernetes**

Google Kubernetes technology in this v1.2.2 version is feature rich and easy to use (*see the following chapter on how to deploy PetClinic sample with it, or the related video*).

This COE provides a ‘Docker Compose compatible YML file to describe the Dockerized application. **Kubernetes drives the deployment of the containers PLUS the configuration of the OpenStack external context**: That’s (*plenty of*) it!



Like Swarm, Kubernetes panel show the API url. But it provides plenty of additional features

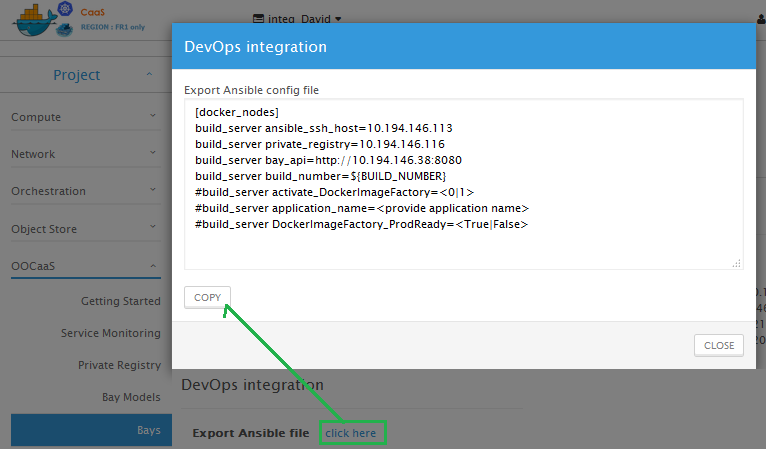
* Kube UI, as a dashboard to manage the deployed container K8S PODs in the cluster
* Kube DNS, as a container service locator to identify the K8S services.
* Kube ELK technology, in order to kepp track of the logs of the running containers
* Kube Collect’D & Graphana for stats on the containers
* Kube configuration inside Magnum-based CaaS offer: K8S encapsulates to use of OpenStack APIs for Storage (see Cinder) and Network (see Neutron/LB and Neutron/FloatingIPs)

In one work: features rich: K8S DO the job for you!

**DevOps chain integration:**  See [devops-and-caas-integration](https://www.dailymotion.com/video/x4b0brh_devops-and-caas-integration_tech) video.

**Retrieving the Bay’s parameters**

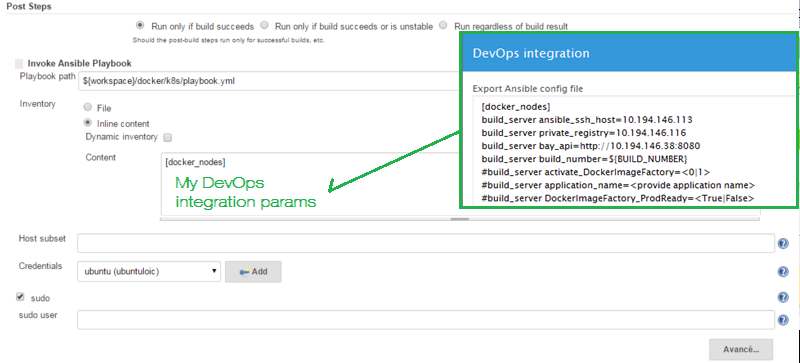
Every bay is providing a ‘*simple but magic*’ dialog Box:



Please click on ‘DevOps integration: the content of the Dialog box corresponds to **the parameters for your future Ansible PlayBook**!

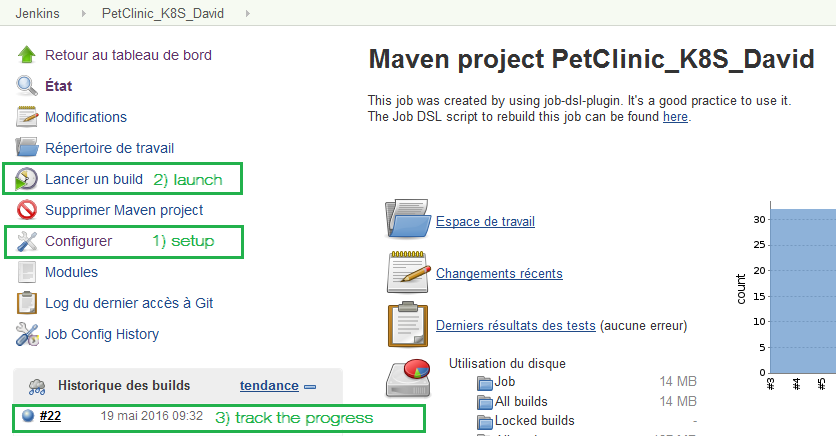
**Configuring your Jenkins job and its Ansible cookbook**

See subset of **Jenkins setup in Annex1 at the end of the document**.   
…***create your new Jenkins job configure it with few technical params, but the Developers are used to this!***



**Building and deploying your first PetClinic application:**  See [deploying-petclinic-in-swarm-bay](https://www.dailymotion.com/video/x4b0bmo_caas-deploying-petclinic-in-swarm-bay_tech) and [deploying-petclinic-in-k8s-bay](https://www.dailymotion.com/video/x4b0auc_caas-deploying-petclinic-in-k8s-bay_tech) videos.

Once you setup the Jenkins job, then launch it and follow the logs progress



As an example, CaaS is providing an example with the PetClinic ‘AllInOne’ Jenkins and related Ansible cookbook is driving:

* The build of the standard Java Spring application, producing a java War file to be onboarded inside a Tomcat web server.
* The build of the ‘PetClinic’ Docker image that will be based on a ‘latest’ tomcat Docker image from the Docker Hub on Internet plus some glue to deploy and configure the PetClinic War
* The tag then push of this image inside the CaaS private registry
* And then the deployment of the application in a Swarm or Kubernetes Bay

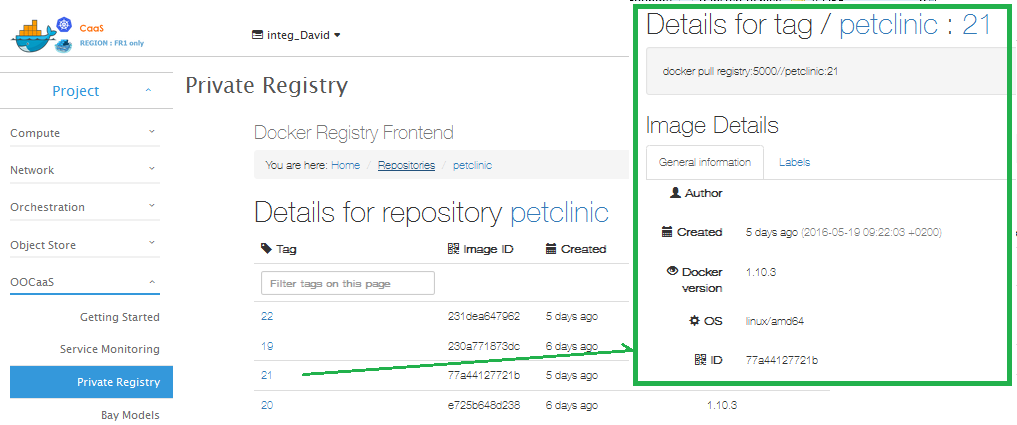
In the following screenshot, the Kubernetes COE is used:

* The first line corresponds to the BuildServer’s request to deploy the PetClinic application in the bay: Kubectl command.
* Then later, when the build is in success, the Kubernetes bay is giving back plenty of info on the features rich content of the bay (already shown in the CaaS ‘Bay’ panel)
* In blue: as a result the URL of the PetClinic service is given back (Developers know the URI of /petclinic ☺ )



**Watch the videos: many more details inside them!!!**

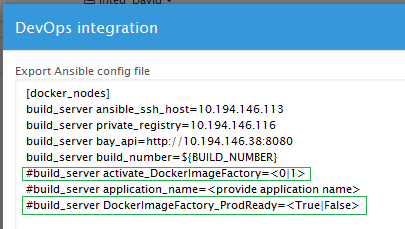
**How to use the Private registry?**

The CaaS UI is providing a ‘Docker dashboard’ integrated as an iFrame. In the Beta Release 1, only Read-only feature is available. Next release will provide Docker ‘Portus’ UI with Read-Write’ functions.  


Please browse the registry: as an example see the PetClinic Docker image with multiple versions (because plenty of Jenkins jobs which tag each time with the Job’s tag). Each version provides some infos.

**How to use the Docker images factory?**

In the DevOps integration dialog box, two attributes are configuring the use (or not) of the embedded Orange image factory framework.

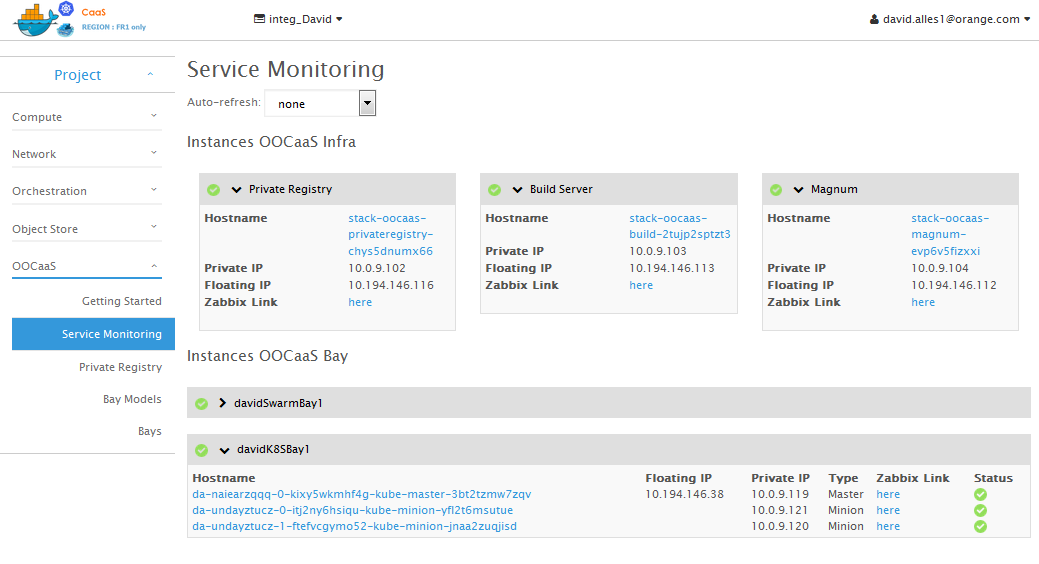


If DockerImageFactory is set to ‘0’, no use of it…

See subset of **Docker image factory setup in Annex2 at the end of the document**.

**How to use the auto-monitoring feature in CaaS with the embedded Zabbix?**

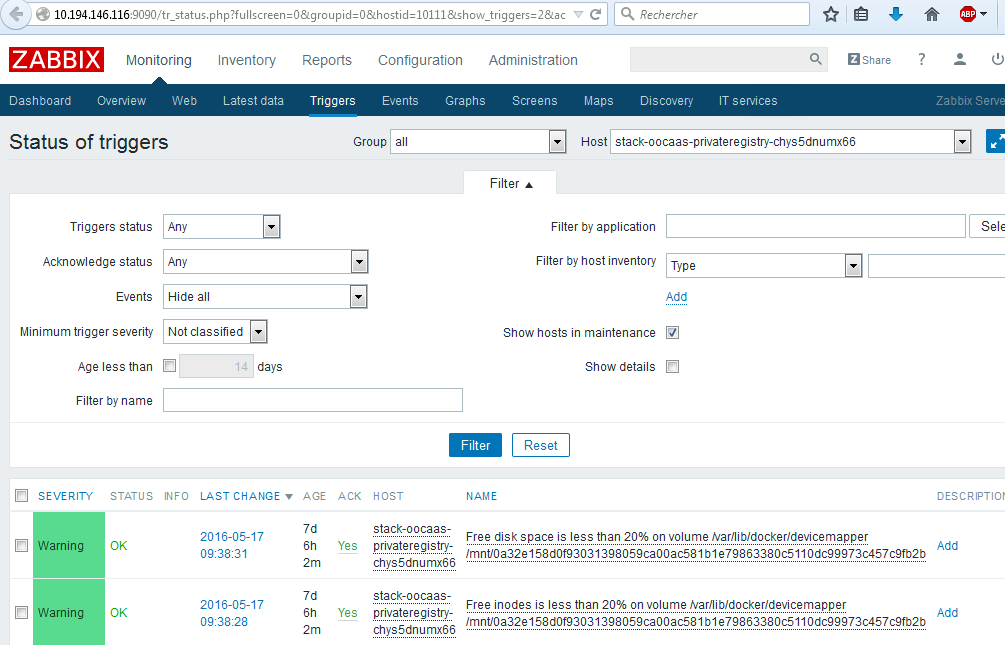
CaaS innovation contains an auto-monitoring feature, see ‘service monitoring’ tab.



By design, every KVM instance deployed inside the customer’s tenant is discovered by the CaaS embedded Zabbix system and assigned to a monitoring template according to their role.

The service monitoring page is classifying the instances into two groups: the ‘CaaS infra’ for the PrivateRegistry, BuildServer and Magnum… and the bays with one entry per bay with their Master and nodes…

When clicking on the related ‘Zabbix link’ on one element, a new tab is open and displays the filtered monitoring element



**And Watt else?**

Next document will detail some more info on

* Understanding the CaaS infrastructure: see [understanding-caas-infrastructure](https://www.dailymotion.com/video/x4cio3y_understanding-caas-infrastructure_tech) video
* How to manage namespaces and repositories inside Docker PrivateRegistry?
* How to fine tune the blacklists in the Docker image factory?
* Operating the dockerized application with Kubernetes focus :
  + How to use Kube UI?
  + How to use Grafana?
  + How to use Kibana?
* Operating the OOCaaS deliverables
  + How to troubleshoot the service?
  + How to use the embedded Zabbix to monitor your application?

**And the future?**

This article will allow you to dive into the Dockerized world on CloudWatt.   
This beta service is currently free of charge for the Docker layer. Please do not hesitate to provide us with your feedback on the current services as well as your ideas for bugFixes, features enhancements or a ‘managed service’ by Orange Business services tm.

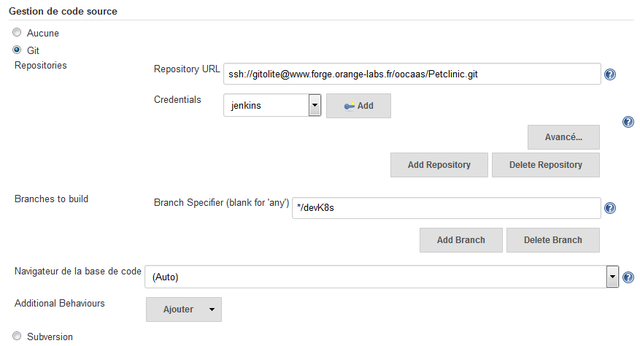
🡺Get in touch with [apps@cloudwatt.com](mailto:apps@cloudwatt.com)

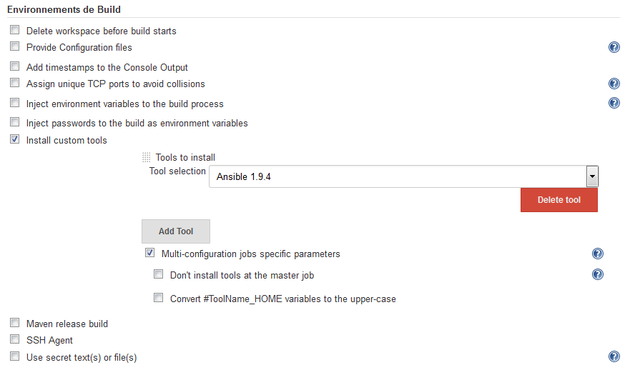
**Annex 1: complete setup of Jenkins and related Ansible**See [devops-and-caas-integration](https://www.dailymotion.com/video/x4b0brh_devops-and-caas-integration_tech) video.

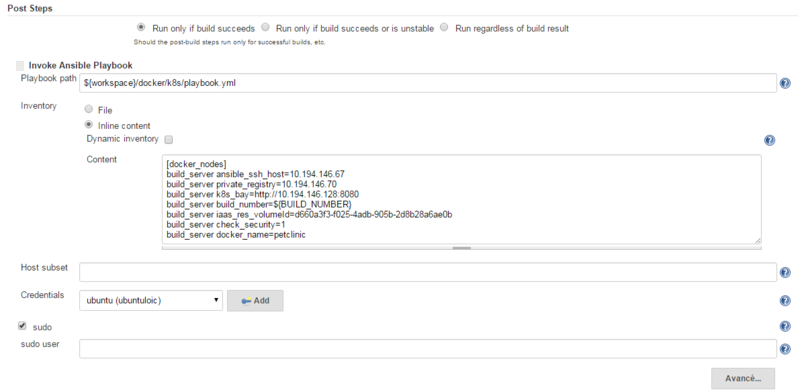
 Add the [Ansible Plugin](https://wiki.jenkins-ci.org/display/JENKINS/Ansible+Plugin) to your Jenkins. (Nb : Read the Jenkins documentation : [How to add a plugin](http://faas.forge.orange-labs.fr/documentation/master/userguide/index.html#jenkins-ajout-plugin))   
[](https://www.forge.orange-labs.fr/plugins/mediawiki/wiki/oocaas/index.php/File:Jenkins_plugin_ansible.png)

 Add the custom tools : Ansible 1.9. (Nb: Read the Jenkis documentation : [How to add a custom tools](http://faas.forge.orange-labs.fr/documentation/master/userguide/index.html#jenkins-custom-tools))   
[](https://www.forge.orange-labs.fr/plugins/mediawiki/wiki/oocaas/index.php/File:Jenkins_custom_ansible.png)

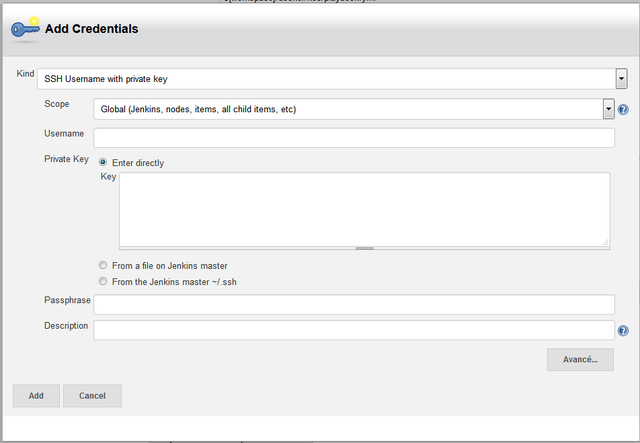
 Create a new Job.

 Configure the job to pull the source code of your application on the Git of your source control.   
[](https://www.forge.orange-labs.fr/plugins/mediawiki/wiki/oocaas/index.php/File:Jenkins_git.png)

 Check the box 'Install custom tools' and select 'Ansible 1.9' as tool selection.   
[](https://www.forge.orange-labs.fr/plugins/mediawiki/wiki/oocaas/index.php/File:Jenkins_ansible.png)

 Add an post build step : 'Invoke Ansible Playbook' and use the exported parameters from your Bay.  
[](https://www.forge.orange-labs.fr/plugins/mediawiki/wiki/oocaas/index.php/File:Jenkins_playbook.png)

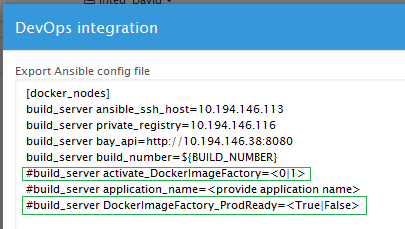
 Configure the step to execute your playbook ansible.

 Add the keypair of the Build Server instance in order that Ansible can connect to it.   
[](https://www.forge.orange-labs.fr/plugins/mediawiki/wiki/oocaas/index.php/File:Jenkins_private_key.png)

**Annex 2: Using the Docker image factory framework**

**Selecting the level of image validation**

In the DevOps integration dialog box, two attributes are configuring the use (or not) of the embedded Orange image factory framework



* If ‘build\_server activate\_DockerImageFactory’ is set to ‘0’, then no image validation is made.  
  As a consequence the build server is only building the docker images and pushing them on the Private Registry
* If ‘build\_server activate\_DockerImageFactory’ is set to ‘1’, then the complementary attribute is use to define with ‘*validation blacklist’* is used:   
   … ‘build\_server DockerImageFactory\_ProdReady =<True|False>’

**Understanding the configuration of the blacklists**

On the CaaS\_BuildServer KVM instance, two files are provided with default Orange values for blacklists:

* ‘Insecure level == **nonProduction** :’ The file can be modified in /home/cloud/imagefactory/run/**filch-insecure.json**
* ‘**Production** level’ : The file can be modified in /home/cloud/imagefactory/run/**filch.json**

{

"packages": {

"authorized": [

"^gcc.\*-base"

],

"forbidden": [

"^gcc",

"^g\\+\\+",

"^firefox",

"^thunderbird",

"^iceweasel",

"^spidermonkey-bin",

"^libmozjs",

"^libsmbclient",

"^libwbclient",

"^samba-common",

"^smbclient",

"^smbfs",

"^samba",

"libpcap",

"iftop",

"wireshark",

"tcpdump",

"^apport",

"build-essential",

"^automake"

]

},

"processes": {

"authorized": [

"ssh-agent"

],

"forbidden": [

"ssh"

]

},

"files": {

"forbidden": [

"tcpdump",

"javac",

"nc",

"hping",

"hping3",

"nmap",

"john",

"unshadow",

"telnet",

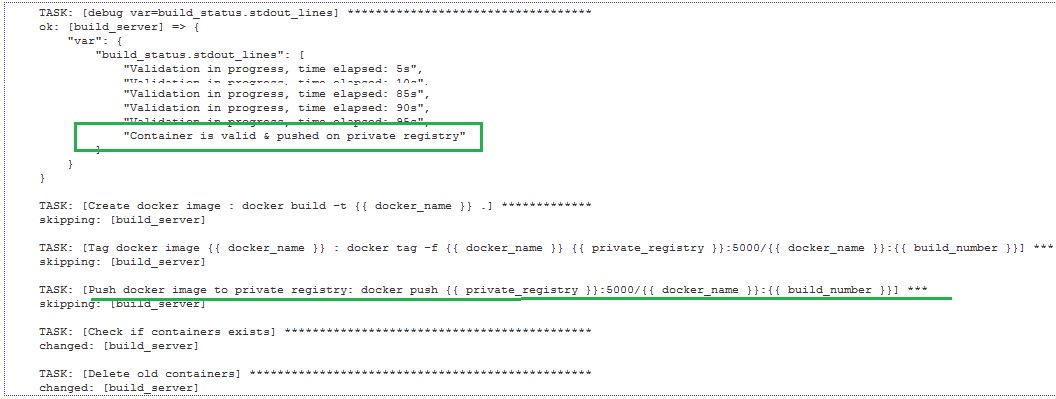
"curl",

"wget"

]

}

**Examples of validation**

* If validation is running well  
  
* If validation is KO  
  