Local Cloud Native Toolkit

setup the Cloud Native Toolkit on a local machine

Brian Innes

None

Table of contents

1. Local Cloud Native Toolkit

This repo is based on the Cloud Native Toolkit. The content of this repository extend the toolkit to run on a pure open source stack, based on minikube.

NOTE: THIS REPOSITORY IS STILL A WORK IN PROGRESS

1.1 Prerequisites

Currently only tested on an Intel based Mac/Linux(Ubuntu) with 16GB of memory or more

This content will currently only run on systems capable of running AMD64 architecture containers, as many of the components are only provided in this architecture

- Docker
- Ansible
- Minikube
- Git Client
- python and pip also need to be installed on the system
- Basic developer tools, to build native components in packages:
- MacOS : working xcode install with additional command line tools (`xcode-select --install`)
- Linux : `sudo apt install -y build-essential`
- Command line tools, kubectl, helm
- Linux Ubuntu : `sudo  snap install --classic helm`

1.2 Setup instructions

install Docker - Ubuntu Linus instructions -

```
snap install docker
sudo groupadd docker
sudo usermod -aG docker $USER
sudo reboot -n
```

install ansible, git, python, pip - Ubuntu - `sudo apt-get install -y ansible, python3-pip`

install minikube - mac instructions - `brew install minikube` - Ubuntu linux instructions

Additional information here

> **Info**
>
> Ideally you want to know the IP address your cluster will be assigned before it is created. On MacOS using hyperkit you can do this by looking at file /var/db/dhcpd_leases. Your machine will be allocated the next available IP address. If the file does not exist or is empty then you will be assigned the first available address on the subnet created for the driver used (hyperkit in the example command below).

1.2.1 Start the Kubernetes cluster

To create the minikube cluster, run the following command, adjusting the CPU and memory settings to match your configuration. You want to give the stack as much memory as possible to get good working performance.

Start minikube. The commands below are recommended minimum resource values. If you have a larger machine you may want to increase CPUs, memory or disk to provide additional resources to minikube: mac : `minikube start --addons=dashboard --addons=ingress --addons=olm --addons=metrics-server --addons=istio-provisioner --addons=istio --cpus=4 --disk-size=50g --memory=12g --embed-certs --driver hyperkit --insecure-registry`

`192.168.64.2:5000` linux : `minikube start --addons=dashboard --addons=ingress --addons=olm --addons=metrics-server --addons=istio-provisioner --addons=istio --cpus=4 --disk-size=50g --memory=12g --embed-certs --driver kvm2 --insecure-registry 192.168.64.2:5000`

Bring up the dashboard with command `minikube dashboard &`, switch to **All namespaces** and then highlight **Workloads** in the side menu.

Wait until all Deployments are complete (Workload status Deployments circle is all green) before continuing.

1.2.2 Fix up insecure registry and domain

Verify the cluster ip address with `minikube ip`.

If you didn't use the IP address returned by the `minikube ip` command for the insecure-registry option in the start command, then you need to complete the instructions in this section. If you were able to use the correct IP address then you can skip to adding the TLS certificates.

Edit file ${HOME}/.minikube/profiles/minikube/config.json and alter the InsecureRegistry value to the ip address returned by the `minikube ip` command.

Do the same with file ~/.minikube/machines/minikube/config.json - ensue the Insecure Registry entry contains the correct IP address.

Restart minikube with `minikube stop` then repeat the start command above, replacing the insecure-registry option to the correct IP address for minikube

> **Info**
>
> In the rest of the documentation it is assumed that minikube is running on IP address 192.168.64.2. Wherever you see that address you need to substitute with the IP address of your cluster.

1.3 Run the ansible installer to complete the setup

In a command window:

- clone this git repository: `git clone https://github.com/binnes/cloud-native-toolkit.git`
- change into the ansible directory: `cd cloud-native-toolkit/ansible`
- ensure required packages are installed `pip3 install cryptography`
- install kubernetes collection `ansible-galaxy collection install community.kubernetes`
- install crypto collection `ansible-galaxy collection install community.crypto`
- install general collection - for docker `ansible-galaxy collection install community.general`
- run the playbook: `ansible-playbook --ask-become-pass minicube-playbook.yml`
- You will be prompted for your user password - this is because some configuration options need admin privileges

At some point you will be prompted for git credentials for the deployed Gitea server. The username and password are defined in the vars.yml file to be demo/dem0P4s$. You can change these if desired.

1.4 Post install steps

A self-signed root certificate has been generated and used to create the ingress TLS certificate. This needs to be trusted by your browser to be able to access the various applications.

The setup script installs the rootCA certificate to the Mac OS keychain, but the command line tool doesn't appear to setup the trust settings correctly. Go into the Keychain app (found in Applications -> Utilities). Select the System keychain and Certificates section. Find the certificate - named *[IP address]*.nip.io-RootCA. Change the trust settings to Never Trust, then change back to Always Trust. When you close the window you should b prompted for your user password, showing that the change has been registered. This certificate will now be trusted by Safari and Chrome.

In Firefox you need to import the certificate. The certificate is located in the .minikube/certs folder within your home folder. (~/.minikube/certs) in a file named rootCA_crt.pem. In Firefox open the Preferences, then select the Privacy & Security settings. Scroll down until you see the certificates section then press the View Certificates button. In the Certificate Manager popup window select the Authorities section then Import... Navigate to the .minikube/certs directory within your home directory and select the rootCA_rt.pem file to import. Select both checkboxes then hit OK to import the certificate.

1.5 Accessing the applications

Minikube runs within a virtual environment on your laptop, so the IP address is not available from outside your system, so by default you need to run on the same machine that minikube is running on. So launch a browser on the machine and navigate to `https://dashboard-tools.[ip address].nip.io`.

```
E.g. if ```minikube ip``` returns 192.168.64.2 then the address will be https://dashboard-tools.192.168.64.2.nip.io.

This is the Cloud Native Developer dashboard.  This has links to the applications the Cloud Native Toolkit installed on your cluster.
```

> **Note**
>
> Directories beginning with a . are usually hidden on MacOS. To see them you can press the **Shift-Command-.** key combination

1.5.1 Gitea setup

> **Info**
>
> If you want to log onto Gitea as an administrator, look in the gitea-init secret in the gitea namespace. There you will see the credentials for the admin user: the username is gitea_admin and the password is also visible.
>
> If you create your own user you can use the admin login to go into the site admin section and promote your user to an administrator.

- Register a new account
- Sign into new account
- Goto settings, then Applications, then Generate Token - copy it and don't loose it
- pipeline : 1dfb4040dd49269740926a512727dbc474fac90b

1.5.2 Setup che

The default user name and password is admin / admin> - need to change password at first login and don't forget what the new admin password is.

[todo - how to create new user account in keycloak]

1.6 Useful commands

1.6.1 Remove all completed pods (successful or failed)

```
kubectl get pods --all-namespaces -o wide | egrep -i 'Error|Completed' | awk '{print $2 " --namespace=" $1}' | xargs kubectl delete pod --force=true --wait=false --grace-period=0
```

1.6.2 manually run the terraform scripts

```
docker run -it -e TF_VAR_server_url="https://`minikube ip`:8443" -e TF_VAR_ingress_hostname="`minikube ip`.nip.io" -e TF_VAR_source_control_url="" -e TF_VAR_ibmcloud_api_key="" -v /Users/brian/Cn-Tk/ibm-garage-iteration-zero/terraform:/home/devops/src -v /Users/brian/.kube/config:/home/devops/.kube/config -v /Users/brian/.minikube:/home/devops/.minikube -w /home/devops/src --name cntk-installer quay.io/ibmgaragecloud/cli-tools:v0.10.0-lite /home/devops/src/runTerraform.sh -k -a
```

1.6.3 Install and run local toolkit

- clone repo
- cd into repo directory .\cloud-native-toolkit\ansible
- `minikube start --addons=dashboard --addons=ingress --addons=olm --addons=metrics-server --addons=istio-provisioner --addons=istio --cpus=8 --disk-size=250g --memory=48g --embed-certs --driver hyperkit --insecure-registry 172.16.171.2:5000`
- `ansible-playbook --ask-become-pass minicube-playbook.yml`

1.6.4 to remove all

- `minikube delete`
- `rm -rf ~/.minikube`
- `rm -rf ~/Cn-Tk`
- `sudo vi /var/db/dhcpd_leases` - remove all content (or just the entry for the cluster)
- `docker ps -a` - remove all exited containers
- remove the certificate 172.16.171.2.nip.io-RootCA from the trusted authorities (Keychain app on Mac)

2. Where to run the Cloud Native Toolkit

The Cloud Native Toolkit is made up of Open Source projects with some helper functions and installation utilities added by the Cloud Native Toolkit project. It has been designed to run on a Kubernetes cluster. Either a generic Kubernetes cluster or an Open Shift cluster. The IBM Cloud is the default installation target.

This project extends the core Cloud Native Toolkit to allow installation on a local laptop or workstation:

2.1 Learning environment

The IBM Cloud managed Open Shift environment is ideal for production use, to support active development teams to develop new applications and services. It can also be used to host learning environments for developers wanting to learn Enterprise Cloud Native development skills. However, there are times when using the Managed OpenShift environment on IBM Cloud may not be ideal, so the chart below shows other options for installing the toolkit.

	OpenShift experience	Air-gapped environment	No cost option	Laptop install
OpenShift on IBM Cloud	YES	NO	NO	NO
OpenShift local install	YES	YES	NO	NO
OKD local install	YES	YES	YES	YES (>= 20GB – min >= 48GB – small)
Code Ready Containers	YES	YES	YES	YES* >= 20GB memory
Minikube / Kubernetes local install	NO	YES	YES	YES >= 16GB memory

2.1.1 OpenShift Experience

OpenShift builds on top of Kubernetes to offer an integrated cloud platform.

There are a specific set of features built into OpenShift for developers, including the Developer perspective, integrated operator hub and private image registry along with the catalog of services and integrated DevOps tooling.

2.1.2 Air gapped environment

Sometimes it is desirable to run disconnected, or where external internet access may not be available, so having the ability to run in an air gapped way allows developers to access the training environment when a cloud hosted environment isn't possible.

2.1.3 No Cost option

Running on a public cloud does incur expense, however, the cost of having to manually install, configure and maintain an environment should not be overlooked when looking at total cost of an environment. However, sometimes a developer wants to create their own private sandbox environment to facilitate deeper learning, needing access permissions to the environment which may not otherwise be available

2.1.4 Laptop install

This is where an isolated environment is needed, either for a personal environment or where there are restrictions in place for internet access.

The key issue with local installs is the resources needed to run an environment. An Enterprise Cloud Native Development environment is designed to be hosted on a cloud, where resources are readily available to support the needs of developers. This scale of resource is not available on a typical developer laptop or workstation.

It is possible to run the cloud native toolkit on a laptop and work through the training material and the chart above shows the options available.

- OKD is the upstream, open source version of OpenShift. It can be run on a number of platforms, including a cut down single host option and a version of Code Ready Containers available on OKD, which is a cut-down, single host install.
- Code Ready Containers (CRC) is a version of OpenShift that is specifically intended to allow developers to have access to a local Open Shift environment. It is bundled as a single virtual machine that runs on the developer laptop and it can run on a 16GB machine, but once additional development tooling is installed and developer activity starts within the CRC environment you quickly run into resource issues, so additional memory is recommended.

> **Warning**
>
> There is currently an issue with the disk resize option on MacOS. The default disk size is sufficient to run OpenShift, but not sufficient to install additional tooling and perform development work within the Open Shift environment.

- Minikube provides a single node, standard Kubernetes installation using minimal system resources, so is ideal for a local kubernetes environment on a laptop. This project extends the Cloud Native Toolkit project to support Minikube as a target install environment. The downside to Minikube is that is is a base Kubernetes install, so the enhanced OpenShift environment is not available to developers, but all the Open Source tooling is still available and many of the integrated features available in OpenShift have an upstream open source project, such as Tekton is the base technology for OpenShift pipelines and Che/Theia is the base technology for Code Ready Workspaces.

There are also some additional considerations when looking at an Air Gapped or local laptop/workstation install, which are discussed here

3. Where to run the Cloud Native Toolkit

The Cloud Native Toolkit is typically run within Open Shift running on the IBM Cloud, however this is not always ideal for learning. Sometime a developer wants a local option, where everything can run on their own hardware, maybe where connectivity isn't always guaranteed. There are other times where a developer may want to dig a little deeper, so wants an isolated environment, where they can experiment without fear of breaking other members of their team, so again want an isolated environment where they have admin permissions.

Having access to a longer term, no cost environment can also give a developer time to do deeper learning

This is why this project exists. To take the Cloud Native toolkit and allow it to be run in an isolated environment with lower resources than would typically be available in a cloud environment.

3.1 Low Resources environment

One of the challenges of working with a local install is that the software stack is designed to run in a cloud environment, where resources are more abundant, and typically designed to serve a large user population. This can make it challenging to replicate on a laptop or developer workstation.

There are also capabilities which are assumed in a cloud environment, which may not exist when running locally, such as inbound connectivity.

This project aims to provide 'as close as is possible' developer experience to the primary target environment on the cloud, but running locally on a laptop or developer workstation.

3.1.1 Minimum Environment

The minimum resources needed to run the toolkit on a laptop or developer workstation is a machine with 16GB memory. The target is to support modern versions of Linux, MacOS and Windows operating systems, but currently only MacOS has been tested

The minimum environment run on top of Minikube rather than OpenShift, as Minikube reduces memory requirements at the cost of the richer cloud platform and the enhanced developer experience provided by OpenShift.

3.1.2 Enhanced Environment

A better developer experience can be obtained where a laptop of developer workstation has more memory available, 24GB or greater. This can then use Code Ready Containers - crc or the Open Source community distribution based version - okd

> **Note**
>
> Use the Cloud Native Toolkit instructions to setup this environment. The project has not yet created the assets to install a stand-alone version of the toolkit based on OpenShift (crc/okd).

3.2 Challenges of running locally

Running the environment locally does pose some challenges to deliver a good developer workflow:

3.2.1 DNS - Name resolution

When you deploy workloads on a cloud your browser and clients of the workloads need to be able to reach the workloads. Facilities like the Ingress controller within the cloud infrastructure will route traffic within the cloud, but a browser still needs to be able to get to the cloud. With public cloud hosted services, the Internet DNS service enables name resolution to an IP address, but such services are not available for local kubernetes clusters, unless you setup and configure your own DNS server or manually configure mappings for all services on your workstation.

To overcome this issue the **nip.io** name resolution service is used by the project. This does require an outbound internet connection to work, but solves name resolution issues for workloads deployed to a local kubrnetes cluster.

The service works by including the cluster IP address in the URL. Service URLs need to be in the form of [service name].[ip address].nip.io and they will resolve the to IP address included in the URL.

This project sets the cluster domain to [minikube ip address].nip.io, so all ingress hostnames will have a valid nip.io format and all will resolve to the kubernetes IP address.

If you need to run in a disconnected environment then a DNS server, such as **dnsmasq** needs to be installed and configured in your local network/workstation.

> **Note**
>
> This is on the todo list to provide instructions to implement local DNS

3.2.2 TLS Certificates

Secure communication is provided by TLS based on certificates and a set of trusted certificate authorities. The public certificates for these trusted certificate issuing authorities are included in operating systems and some browsers. When working with services accessible over this internet then you can configure a cluster to dynamically request certificates or purchase certificates for use within an enterprise. However, to use free certificate authorities, such as LetsEncrypt you need to verify you control the domain you are requesting a certificate for, so need to setup a server to accept incoming requests for that domain, as registered in public DNS servers.

With a local setup it is not always possible to set this up and where it is possible it is often not trivial. Having to use a traffic forwarding service, dynamic DNS forwarder, etc. then having to configure your service provider router or modem to forward traffic within your local network. If working at a hotspot or with certain internet service providers it is not possible to accept inbound traffic, so an alternate solution is needed to provide SSL/TLS certificates.

This project creates a self-signed root Certificate Authority (CA) certificate, then issues a wildcard server certificate to the cluster *[minikube IP address].nip.io, which is signed by the self-signed root CA certificate. You need to add and trust the public certificate of the self-signed root CA to your OS/browser then all services offered by the local cluster will be trusted and you won't receive any browser warnings. Any application or services that need to access a cluster service will also need to have the root CA public certificate added to their host OS to allow them to communicate securely without reporting certificate authorisation errors.

This approach works for disconnected or connected working, once the CA root public certificate is trusted.

3.2.3 Inbound connectivity to trigger pipelines

The Cloud Native Toolkit relies heavily on the source control system, usually github. However, part of the workflow requires git hooks to fire and initiate actions within the development environment. When running on public cloud infrastructure github.com is able to connect with the required services running the developer workflow, but in a local environment this is not possible.

To provide the integrated developer workflow this project installs a private git service on the cluster, so git hooks can be configured to initiate workflows within the local cluster.

The current choice of private git service is Gitea, but the Cloud Native Toolkit has just added support for Gogs, so it may be preferable to adopt that in the future?

4. Running the Day One instructions

This page outlines the additional steps needed to run through the day 1 instructions

4.1 Setup additional steps - before running the oc sync command

Currently the setup doesn't fully create all the needed kubernetes objects.

You need to update and then apply the following configuration files in the kubernetes folder of this repository. The only updates needed are to fix up the IP address. This should be the IP address returned by the `minikube ip` command :

- ibmcloud-config.yaml
- registry-config.yaml

To apply the files simply user the kubectl or oc command:

```
kubectl apply -f ibmcloud-config.yaml
kubectl apply -f registry-config.yaml
```

4.2 Fixing up the Tekton Pipeline configuration

The command line tool doesn't create all the required tekton configuration or the Gitea webhook. In addition some of the created pipeline configuration assumes OpenShift running on the IBM Cloud, so need to be updated to run locally.

1. Apply the following 3 files. You need to update the content to deploy the configuraion to your project namespace (bi-dev in the files in the repo kubernetes folder)

```
kubectl apply -f tekton-gitea-clusterBinding.yaml
kubectl apply -f test-event-listener.yaml
kubectl apply -f test-eventListener-ingress.yaml
```

2. The generated TriggerTemplate for the project is not correct. the params path to retrieve the git configuration values needs to be tt.params rather than just params. To fix this use the Kubernetes dashboard app or command line to update the Custom Resource Definition for the triggertemplates object created for the project. An additional config property also needs to be added to allow the ingress type to be overridden. By default an OpenShift route will be created rather than a kubernetes route. The params section should be as seen below:

```
spec:
  params:
    - name: git-url
      value: $(tt.params.gitrepositoryurl)
    - name: git-revision
      value: $(tt.params.gitrevision)
    - name: scan-image
      value: 'false'
    - name: deploy-ingress-type
      value: ingress
```

2. Again in the TriggerTemplate object for the project the ServiceAccount *pipeline* should be used to run tasks, as this serivce account has the necessary roles assigned. Using the default account causes RBAC access failures. To add the service account modify the TriggerTemplate object for the project and set the serviceAccountName property:

```
resourcetemplates:
  - apiVersion: tekton.dev/v1beta1
    kind: PipelineRun
    metadata:
      generateName: stockbffnode-
    spec:
      params:
        - name: git-url
          value: $(tt.params.gitrepositoryurl)
        - name: git-revision
          value: $(tt.params.gitrevision)
        - name: scan-image
          value: 'false'
        - name: deploy-ingress-type
          value: ingress
      pipelineRef:
        name: stockbffnode
      serviceAccountName: pipeline
```

3. The role definition for the pipeline needs to be adjusted as the ingresses resource belongs to the *networking.k8s.io* API Group, not extensions. The role definition within your project namespace should be:

```
kind: Role
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: pipeline
  namespace: bi-dev
rules:
  - verbs:
      - '*'
    apiGroups:
      - ''
    resources:
      - services
      - pods
      - pods/exec
      - pods/log
      - secrets
      - configmaps
  - verbs:
      - '*'
    apiGroups:
      - apps
    resources:
      - deployments
  - verbs:
      - '*'
    apiGroups:
      - networking.k8s.io
    resources:
      - ingresses
```

4. The Tekton Pipeline definition needs to be updated to pass the deploy-ingress-type parameter to the tasks. Update the Custom Resource Definition for the Pipeline object created for the project. The project pipeline object should be changed to include :

```
spec:
  params:
    - description: The url for the git repository
      name: git-url
      type: string
    - default: master
      description: 'The git revision (branch, tag, or sha) that should be built'
      name: git-revision
      type: string
    - default: 'true'
      description: Flag indicating that an image scan should be performed
      name: scan-image
      type: string
    - default: ingress
      description: Create an OpenShift route or Kubernetes Ingress configuration
      name: deploy-ingress-type
      type: string
  tasks:
    - name: setup
      params:
        - name: git-url
          value: $(params.git-url)
        - name: git-revision
          value: $(params.git-revision)
        - name: scan-image
          value: $(params.scan-image)
        - name: deploy-ingress-type
          value: $(params.deploy-ingress-type)
      taskRef:
        kind: Task
        name: ibm-setup
```