

Multi Node Kubernetes developer environment - using VBox /Vagrant/CoreOS

Features of an Effective a Developer Environment

- Convenience and Productive
 - Fast/Easy to install - everthing for development included by default
 - Fast/Easy to start/stop/update/teardown
 - Ability to reproduce and share environments
 - Easy to manipulate multiple instances/environments and operate simultaneously
 - (golden/dev/test/experimental), compare/contrast/diff environments
 - experiment with federation and multi-site deployments in DevEnv
 - Export/Import/shareable instances - reproducible in many contexts
 - Share with others, publish, reproducible environment as part of bug reports, ...
- Reliability and Isolation
 - Instances isolated from each other and the developers system and from on-line requirements
 - Encapsulated - full environment enclosed in the developer environment
 - Run multiple clusters in a laptop in an airplane disconnected from internet
 - Intentional upgrade process - avoid breaking auto-update
 - (not automated), encapsulated (minimal dependencies on external system/tools)
- Accuracy and Portability
 - Represents the 'production' implementation of the system as closely as possible
 - Run in all popular developer laptops/system/OS combinations
 - Windows, Linux, Mac, ...
 - Developments in DevEnv are directly applicable to production systems
 - Support development at all-levels as closely as possible
 - system-level, operations-level, apps, user-facing, and development at all levels
- Extensibility - Ability to extend and customize the platform as well as the developer environment
 - I.e. no specific tool or model lock-in, freedom to change
 - Network - emulate multiple physical networks, ability to change k8s network plugins
 - Distributed - Emulate multi-site, multi-instance in dev
 - Federation, Distributed/federated/multi-site/edge-oriented development and testing

Some Developer Use Cases

- Development of WB API/Web services
- Development of integration with external storage/compute - DataDNS
- Experimentation with K8s Federation - Federating multiple clusters and and A/B (reference/experimental) dev/testing in a developer environment
- Experimentation/development of in-cluster system services: services in k8s that support NDS-level services
 - gluster automation, PVCs and alternate storage/volume models in WB, in-cluster authz/authn, network policy security, ...
- Test/Evaluation of systems-level SW: New k8s versions, different container runtimes/plugins, different network overlay plugins

5 minutes to a running multi-node cluster that can run in your laptop. VBox machines running CoreOS on host-local networking, not exposed to internet. Notes below for other options.

Here is the recipe, adapted from the official instructions here: <https://coreos.com/kubernetes/docs/latest/kubernetes-on-vagrant.html>

1. Install VirtualBox
2. Install Vagrant
 - a. Check vagrant: "vagrant box list"
3. Install kubectl
 - a. curl -O <https://storage.googleapis.com/kubernetes-release/release/v1.5.2/bin/linux/amd64/kubectl>
 - b. install in /usr/local/bin or your favorite spot in your PATH
4. git clone <https://github.com/coreos/coreos-kubernetes.git>
5. cd multinode/vagrant
6. vagrant box add http://alpha.release.core-os.net/amd64-usr/current/coreos_production_virtualbox.json
 - a. will download CoreOS for virtualbox
7. cp config.rb.sample to config.rb and edit:
 - a. Pick the channel and node sizes and counts according to your laptop resources (primarily memory), here's one example:

```
$update_channel="alpha"

$controller_count=1

$controller_vm_memory=512

$worker_count=2

$worker_vm_memory=1024

$etcd_count=1

$etcd_vm_memory=512
```

8. Good to check your networking for collisions on unrouted address ranges that you'll encounter at work, home, etc.

- a. The default VM network specs are in the Vagrantfile: 172.17.4/24
9. `vagrant up --provider virtualbox`
10. `export KUBECONFIG="${KUBECONFIG}:(pwd)/kubeconfig"`
 - a. Alt: Add the config to any existing `~/.kube/config` if you are working with multiple clusters and or namespaces (`kubectl set-cluster`, `kubectl set-context`, etc.)
11. Should be operational, some useful commands (use in the `multinode/vagrant` path)
 - a. `vagrant ssh c1` (or the other names) to get into coreos
 - b. `vagrant status`
 - c. `vagrant suspend/resume`
 - i. Once in a while net doesn't come back, and `kubectl` says:
 The connection to the server 172.17.4.101:443 was refused - did you specify the right host or port?
 - ii. In this case you can `vagrant halt`, `vagrant up`
 - d. `vagrant destroy` (obliterates)
 - i. Just `vagrant up` and restart afresh

Comparison to Hyperkube (container-based) local install

Advantages to local container options:

DevSystem	Advantage	Impact	Notes
Hyperkube	Single dependency	Low	docker install vs Vagrant and VirtualBox
	Efficiency	Low for development class systems	Container-based is more efficient in terms of resources. VM's are always overprovisioned
Vagrant	Familiarity with process	Moderate	Use of VM's by developers is a common development model/process that's widely understood Known familiar model is easy to adopt and faster to get started - more self-serving w.r.t. problems.
	Isolation between instances and from developer system: better stability, reproducibility, easier support	High	No entertwining with local docker environment - avoids widespread docker troubles/inconsistencies/breaking updates easier cleanup/reinit of environments without local docker cleanups ability to run more than one environment Overloading Docker will not kill developers system - observed problem in NDS Avoids potential collision of WB images and other local docker images unrelated to WB devenv.
	More accurate w.r. t. hosted and self-hosted production environments	Low - App Development High - Systems Development	True emulation of multi-machine deployment, true multi-network capabilities with routing, true ingress without hostport collision, true operations emulation of machine failure/upgrade/maintenance, ability to work with node device-layer storage.
	Flexibility	Low - App Development High - Systems Development	Changeout capabilities: network layer, docker layer, OS layer Can experiment with different OS and/or multiple OS on different machines in single cluster Complex developments can be exported/shared and disseminated for evaluation and/or group development purposes. Accelerates development in comparison to the starting-from-ground-zero/replay model and allows more involved experimentation with complex environments. Can support addition non-cluster vm's with tooling/ops environments - like the gcloud shell Supports low-level OS and networking debug tools that are
	Supportability	Very High and potential non-value-add timesink	Uniform support through VM image gives full intentional control of configurations and upgrades - this is a huge ongoing issue in the k8s community for a reason (not simply an issue of opinionation) docker change has been an enormous support timesink - and NDS has direct experience on multiple occasions with this - avoid docker update dependency if possible. Self-test by devs having trouble: stop the trouble env, start a new one to test basic functionality and A/B test to discover where things went wrong State of devenv is transferrable - can checkpoint the VMs and share the state with support to reproduce troublesome/important issues - yes semi large-files (but compare to jupyter pull?), but easy in comparison to trying to share a hyperkube environment.

Notes:

1. Can be used for volume/fs work by adding VBox disks to nodes. Kubernetes doesn't have a VBox/vagrant volume driver (yet), so you get raw disk on nodes like we have in OpenStack.

2. Can be used for ingress/network by adding additional interface to the node to be LB.
3. It can be left running, or vagrant suspend/resume can freeze/thaw the whole cluster in a few seconds if you want it toiling around in the background. Can also vagrant halt/up to shutdown/reboot and cluster state (etcd) is preserved.